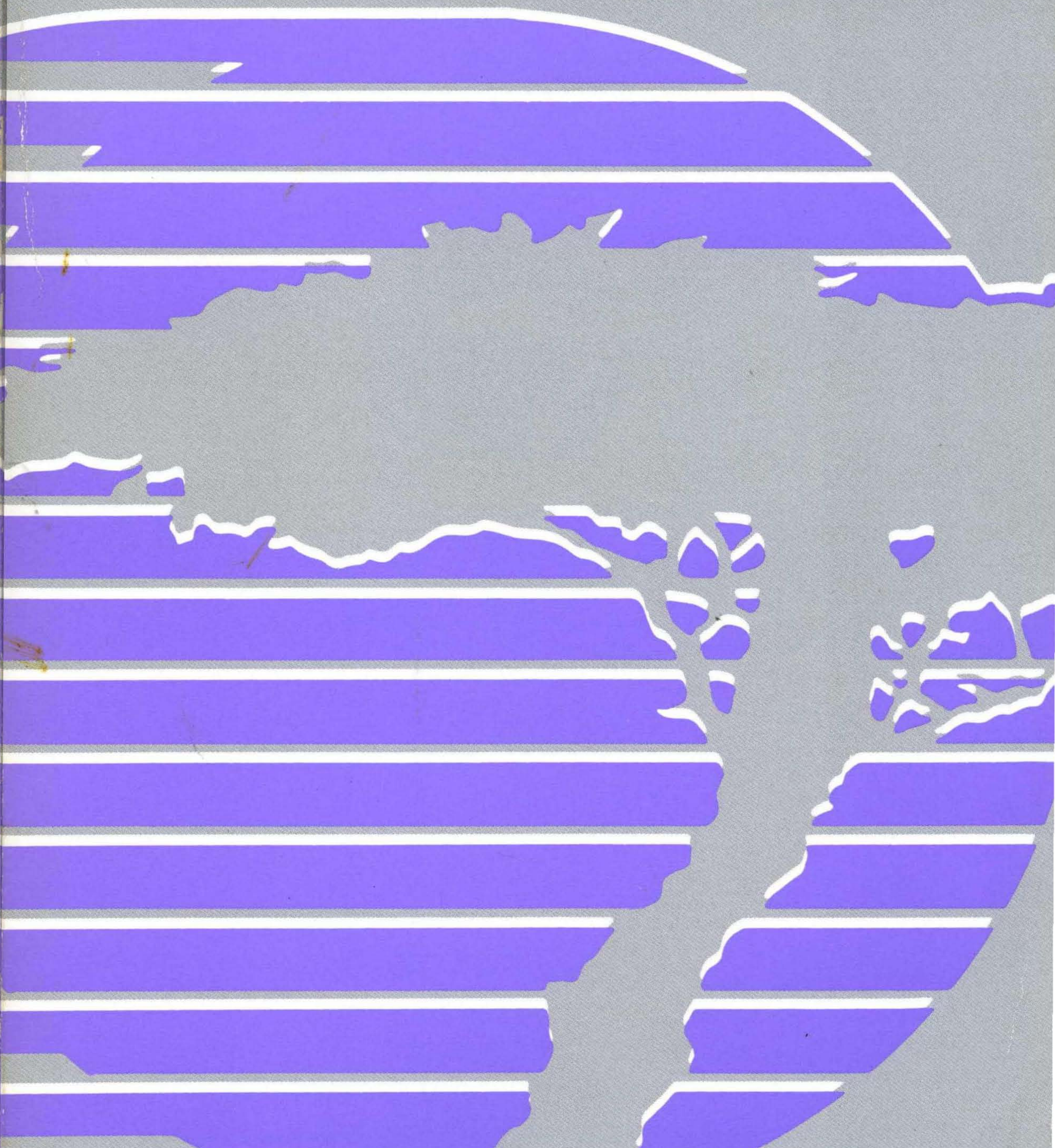
APPLICATIONS HANDBOOK

CYPRESS SEMICONDUCTOR



APPLICATIONS HANDBOOK

CYPRESS SEMICONDUCTOR

APPLICATIONS HANDBOOK

CYPRESS
SEMICONDUCTOR ™

Cypress Semiconductor, 3901 North First St., San Jose, CA 95134 (408) 943–2600

Telex: 821032 CYPRESS SNJ UD, TWX: 910 997 0753, FAX: (408) 943–2741

How to Use This Book

This book has been organized by product type, beginning with general articles that apply to all Cypress products. The individual applications notes follow, organized by product type. The order is: SRAMs, PROMs, EPLDs, Logic (including FIFOs and dual port RAMs), and RISC. Within each chapter, application notes are arranged in the order of part number. In cases where more than one Cypress product is used in the application, the article will be filed using the product which is the primary focus of the article.

Published August 1, 1989

Table of Contents

RISC Page Number

Section Contents

CYPRESS SEMICONDUCTOR

Systems Design Considerations When Using Cypress CMOS Circuits

Introduction

This document is intended to be a guide for the systems designer. Its purpose is to make him aware of the things to consider either when designing new systems using Cypress high performance CMOS integrated circuits or when Cypress products replace either bipolar or NMOS circuits in existing systems. The two major areas of concern are transmission line effects due to impedance mismatching between the source and load, and device input sensitivity.

Design for Performance

In order to achieve maximum performance when using Cypress CMOS integrated circuits, the systems designer must pay attention to the placement of the components on the Printed Circuit Board (PCB), the routing of the metal traces that interconnect the components, the layout and decoupling of the power distribution system on the PCB and, perhaps most important of all, the impedance matching of (some of) the traces (which, under certain conditions, must be analyzed as transmission lines) between the source and the loads. The most critical traces are those of clocks, write strobes (on SRAMS), and chip enables.

Issues of Concern When Cypress ICs Replace Either Bipolar or NMOS ICs

Cypress CMOS ICs have been designed to replace both bipolar ICs and NMOS products, and to achieve equal or better performance at one-third (or less) the power of the components they replace.

When high performance Cypress CMOS circuits replace either bipolar or NMOS circuits in existing sockets, the user must be aware of certain conditions, which may be present in the existing system, that could cause the Cypress ICs to behave in a manner different than expected. These conditions fall into two general categories; (1) device input sensitivity and, (2) sensitivity to reflected voltages.

Input Sensitivity

High performance products, by definition, require less energy at their inputs to change state, than low or medium performance products.

Unlike a bipolar transistor, which is a current sensing device, a MOS transistor is a voltage sensing device. In fact, a MOS circuit design parameter called 'K' is analogous to the gm of a vacuum tube, and is inversely proportional to the gate oxide thickness.

The thin gate oxides, which are required to achieve the desired performance, result in highly sensitive inputs that require very little energy. High frequency signals that bipolar devices would not respond to may be detected by CMOS products.

MOS transistors also have extremely high (5 to 10 million ohm) input impedances, which make their gate inputs analogous to the input of a high gain amplifier (or an RF antenna). In contrast, bipolar ICs have input impedances of 1000Ω or less, so they require much more energy to change state than MOS ICs. In fact, a Cypress IC requires less than 10 picojoules of energy to change state.

Therefore, when Cypress CMOS ICs replace either bipolar or NMOS ICs in existing systems, they may respond to pulses of energy that are present in the system that are not detected by the bipolar or NMOS products.

Reflected Voltages

Cypress CMOS ICs have very high input impedances and, to achieve TTL compatibility and to drive capacitive loads, low output impedances. The impedance mismatch, due to low impedance outputs driving high impedance inputs may, under certain conditions, cause unwanted voltage reflections and ringing, which could result in less than optimum system operation.

When the impedance mismatch is very large, a nearly equal and opposite negative pulse is reflected back from the load to the source when the (electrical) length of the line (PCB trace) is greater than

$$\ell = \frac{T_R \text{ (ns)}}{2\,T_{pd} \text{ (ns/ft.)}}$$

where T_R is the rise time of the signal at the source and T_{pd} is the one-way propagation delay of the line per unit length.

The input clamping diodes that bipolar logic "IC families" (e.g., TTL, LS, ALS, FAST) all have are inherent in the fabrication process. The p-substrate is usually grounded and n wells are used for the NPN transistors and p type resistors. The wells are reverse biased by connecting them to the V_{CC} supply. As a result, a PN junction diode is formed between every input pin (cathode or n material)

December 1986

Introduction (Continued)

and the substrate (anode or p-material). When a negative voltage occurs at an input pin, either due to lead inductance or to a voltage reflection, the diode is forward biased, turns on, and clamps the input pin to a Vf below ground (approximately $-0.8V$).

As circuit performance improved, the output rise and fall times of the bipolar circuits decreased to the point where voltage reflections began to occur (even for short traces) when there was an impedance mismatch between the line and the load. Most users, however, were unaware of these reflections because they were suppressed by the clamping action of the diodes.

Conventional CMOS processing results in PN junction diodes. However, they adversely affect the ESD (Electrostatic Discharge) protection circuitry at each input pin and cause an increased susceptibility to latchup. To eliminate this, a substrate bias generator is used.

Voltage reflections should be eliminated by using impedance matching techniques and crosstalk should be reduced by careful PCB layout.

Crosstalk

The rise and fall times of the waveforms generated by the output circuits are 2 to 4 ns between levels of 0.4V and 4V. The fast transition times and the large voltage swings could cause capacitive and inductive coupling (crosstalk) between signals if insufficient attention is paid to PCB layout. Crosstalk is reduced by avoiding running PCB traces parallel to each other. If this is not possible, ground traces should be run between signal traces. In synchronous systems, the worst time for the crosstalk to occur is during the clock edge with which the data is sampled. In most systems it is sufficient to isolate the clock and other data strobe lines so that they do not cause coupling to the data lines.

The Theory of Transmission Lines

A connection (trace) on a PCB should be considered as a transmission line if the wavelength of the applied frequency is short compared to the line length. If the wavelength of the applied frequency is long compared to the length of the line, conventional circuit analysis can be used.

In practice, transmission lines on PCBs are designed to be as nearly lossless as possible. As a result, the mathematics required for their analysis, compared to a lossy (resistive) line can be simplified.

Ideally, all signals between ICs travel over constant-impedance transmission lines that are terminated in their characteristic impedances at the load. In practice this ideal situation is seldom achieved for a variety of reasons.

Perhaps the most basic reason is that the characteristic impedances of all real transmission lines are not constants, but present different impedances depending upon the frequency of the applied signal. For "classical" transmission lines driven by a single frequency signal source the characteristic impedance is "more constant" than when the transmission line is driven by a square wave or a pulse.

A square wave is composed of an infinite set (Fourier series expansion) of discrete frequency components, i.e., fundamental plus odd harmonics of decreasing amplitudes. When the square wave is propagated down a transmission line the higher frequencies are attenuated more than the lower frequencies and, due to dispersion, all of the frequencies do not travel at the same speed.

Dispersion indicates the dependance of phase velocity upon the applied frequency. (Ref. 1, pg. 192). The result is that the square wave is distorted when all of the frequency components are added together at the load.

A secondary reason why practical transmission lines are not ideal is that they frequently (of necessity) have multiple loads. The loads may be distributed along the line at regular (or irregular) intervals or they may be lumped together (as close as practical) at the end of the line. The signal-line reflections and ringing caused by impedance mismatches, nonuniform transmission line impedances, inductive leads, and non-ideal resistors could compromise the dynamic system noise margins and cause inadvertent switching.

One of the system design objectives is to analyze the critical signal paths and design the interconnections such that adequate system noise margins are maintained. There will always be signal overshoot and undershoot. The objective is to accurately predict them and to keep them within acceptable limits.

The Ideal (Lossless) Transmission Line

An equivalent circuit for a transmission line is presented in *Figure 2.1.* It consists of subsections of series resistance (R) and inductance (L) and parallel capacitance (C) and shunt admittance (G) (or parallel resistance, R_P). For clarity and consistency these parameters will be defined per unit length. The value of the parameter (R, L, C, R_P) must be multiplied by the length of the subsection, ℓ, to find the total value. The line is assumed to be infinitely long.

If the line of *Figure 2.1* is assumed to lossless ($R = 0$, $R_P = \text{infinity}$) *Figure 2.1* is reduced to Figure 2.2.

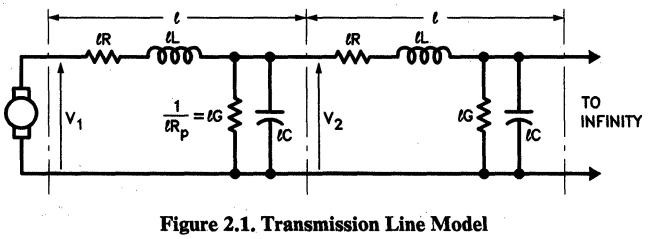


Figure 2.1. Transmission Line Model

The Theory of Transmission Lines (Continued)

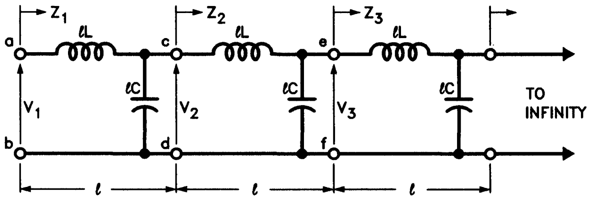


Figure 2.2. Ideal Transmission Line Model

Input or Characteristic Impedance

We shall now calculate the characteristic impedance (AC impedance or surge impedance) looking into terminals a-b of Figure 2.2.

Let the input impedance looking into terminals a-b be Z1, that looking into terminals c-d be Z2, that looking into terminals e-f be Z3, etc. The input impedance, Z1, looking into terminals a-b is the series impedance of the first inductor (ℓ L) in series with the parallel combination of Z2 and the impedance of the capacitor (ℓ C).

From AC theory:

$$XL = j\omega \ell L$$

Where XL is the inductive reactance.

$$XC = \frac{1}{j\omega \ell C}$$

Where XC is the capacitive reactance.

Then $$Z1 = XL + \frac{Z2\,XC}{Z2 + XC} \qquad (2\text{-}1)$$

If the line is "reasonably" long Z1 = Z2 = Z3. Substituting Z1 = Z2 into equation 2-1 yields;

$$Z1 = XL + \frac{Z1\,XC}{Z1 + XC}.$$

Or, $$Z1^2 - Z1\,XL - XC\,XL = 0 \qquad (2\text{-}2)$$

Substituting the expressions for XC and XL yields;

$$Z1^2 - j\omega \ell L = \frac{L}{C} \qquad (2\text{-}3)$$

Equation 2-3 contains a complex component that is frequency dependent. It can be eliminated by allowing ℓ to become very small and by recognizing that the ratio L/C is constant and independent of ℓ or ω.

$$Z1 = \sqrt{\frac{L}{C}} \qquad (2\text{-}4)$$

The AC input impedance of a purely reactive, uniform, lossless line is a resistance. This is true for AC or DC excitation.

Propagation Velocity and Propagation Delay

The propagation velocity (or phase velocity) of a sinusoid traveling on an ideal line (Ref. 1, pg. 33) is:

$$\alpha = \frac{1}{\sqrt{LC}}.$$

The propagation delay for a lossless line is the reciprocal of the propagation velocity.

$$T_{pd} = \sqrt{LC} \qquad (2\text{-}5)$$
$$= Z1\,C$$

where L and C are the intrinsic line inductance and capacitance per unit length.

If additional stubs or loads are added to the line the propagation delay will increase by the factor (Ref. 2, pg. 129).

$$\sqrt{1 + \frac{C_D}{C}}.$$

Where C_D = load capacitance.

Therefore, the propagation delay, T_{PD}', of a loaded line is:

$$T_{PD}' = T_{PD}\sqrt{1 + \frac{C_D}{C}}. \qquad (2\text{-}6)$$

The characteristic impedance of a capacitively loaded line is decreased by the same factor that the propagation delay is increased.

$$Z1' = \frac{Z1}{\sqrt{1 + \frac{C_D}{C}}} \qquad (2\text{-}7)$$

Reflection Coefficients

The third attribute of the ideal transmission line; reflection coefficients, are not actually a line characteristic. The line is treated as a circuit component (which it is) and reflection coefficients are defined that measure the impedance mismatches between the line and its source and the line and its load. The reason for defining the reflection coefficients will become apparent later when it will be shown that if the impedance mismatch is sufficiently large, either a negative voltage or a positive voltage may be reflected back from the load to the source, where it may either add to or subtract from the original signal. If the impedance of the source is mismatched to the line impedance it may also cause a voltage reflection, which in turn will be reflected back to the load. Therefore, two reflection coefficients will be defined.

For classical transmission lines driven by a single frequency source the impedance mismatches cause standing waves. When pulses are transmitted and the output impedance of the source changes depending upon whether a LOW to HIGH or a HIGH to LOW transition occurs, the analysis is further complicated. Classical transmission analysis,

The Theory of Transmission Lines (Continued)

where pulses are represented by complex variables with exponentials, could be used to calculate the voltages at the source and the load after several back and forth reflections. However, these complex equations tend to obscure what is physically happening.

Energy Considerations

Consider next, driving the ideal transmission line from a source capable of generating digital pulses and analyze the behavior of the line under various driving and loading conditions.

The circuit to be analyzed is illustrated in *Figure 2.3.* The ideal transmission line of length ℓ is being driven by a digital source of internal resistance R_S and loaded with a resistive load of RL. The characteristic impedance of the line appears as a pure resistance, $Z_O = \sqrt{L/C}$ to any excitation.

The ideal case is when $R_S = Z_O = RL$. The maximum energy transfer from source to load occurs under this condition, and there are no reflections. One half the energy is dissipated in the source resistance, R_S, and the other half is dissipated in the load resistance, RL, (the line is lossless).

If the load resistor is greater (larger) than the characteristic impedance of the line there will be extra energy available at the load, which will be reflected back to the source. This is called the underdamped condition, because the load underuses the energy available. If the load resistor is smaller than the line impedance the load will attempt to dissipate more energy than is available. Since this is not possible, a reflection will occur that is a signal to the source to send more energy. This is called the overdamped condition. Both of these cases will cause negative traveling waves, which would cause standing waves if the excitation were sinusoidal. The condition $Z_O = RL$ is called critically damped.

It should be intuitively obvious to the reader that the "safest" termination condition, from a systems design viewpoint, is the slightly overdamped condition. No energy is reflected back to the source.

Derivation of the Line Voltage for Step Function Excitation

The procedure is to apply a step function to the ideal line and to analyze the behavior of the line under various loading conditions. The following section will analyze pulses, reflections from various terminations, and the effects of rise times on the waveforms.

The step function response is important because any pulse can be represented by the superposition of a positive step function and a negative step function, delayed in time with respect to each other. By proper superposition the response of any line and load to any width pulse can be predicted. The principle of superposition applies to all linear systems.

According to theory, the risetime of the signal driven by the source is not affected by the characteristics of the line. This has been substantiated in practice by using a special coaxially constructed reed delay that delivered a pulse of 18 amperes into 50Ω with a risetime of 0.070 ns (70 ps). (Ref. 1, pg. 162).

The equation representing the voltage waveform going down the line (Figure 2.3) as a function of distance and time is:

$$VL(X, t) = VA(t)\,U(t - X\,t_{pd}) \quad \text{for } t < T_O \qquad (2\text{-}8)$$

Where:
$$VA(t) = V_S(t)\left(\frac{Z_O}{Z_O + R_S}\right) \qquad (2\text{-}9)$$

VA = the voltage at point A

X = the voltage at a point X on the line

ℓ = the total line length

t_{pd} = the propagation delay of the line in ns/ft.

$T_O = \ell\, t_{pd}$, or the one-way line propagation delay

$U(t)$ = a unit step function occurring at $X = 0$, and

$V_S(t)$ = the source voltage

When the incident voltage reaches the end of the line a reflected voltage, VL′, will occur if RL is not equal to Z_O. The reflection coefficient at the load, ρL, can be obtained by applying Ohm's Law.

The voltage at the load is $VL + VL'$, which must be equal to $(I_L + I_L')RL$. But $I_L = VL/Z_O$ and $I_L' = -VL'/Z_O$ (the minus sign is due to I_L being negative. i.e., it is opposite to the current due to VL.)

Therefore,

$$VB = VL + VL' = \left(\frac{VL}{Z_O} - \frac{VL'}{Z_O}\right)RL \qquad (2\text{-}10)$$

By definition:

$$\rho L = \frac{\text{reflected voltage}}{\text{incident voltage}} = \frac{VL'}{VL}.$$

Solving for VL′/VL in equation 2-10 and substituting in the equation for ρL yields:

$$\rho L = \frac{RL - Z_O}{RL + Z_O}. \qquad (2\text{-}11)$$

The reflection coefficient at the source is:

$$\rho S = \frac{R_S - Z_O}{RL + Z_O}. \qquad (2\text{-}12)$$

Re-arranging equation 2-10 yields:

$$VB = VL + VL' = \left(1 + \frac{VL'}{VL}\right)VL = (1 + \rho L)\,VL \quad (2\text{-}13)$$

Equation 2-13 describes the voltage at the load (VB) as the sum of an incident voltage (VL) and a reflected voltage (ρL VL) at time $t = T_O$. When $RL = Z_O$ no voltage is reflected. When $RL < Z_O$ the reflection coefficient at the load is negative, so the reflected voltage subtracts from the incident voltage, giving the load voltage. When $RL > Z_O$ the reflection coefficient is positive, so the reflected voltage adds to the incident voltage, again giving the load voltage. Note that the reflected voltage at the load has been defined as positive when traveling toward the source. This means that the corresponding current must be negative, subtracting from the current driven by the source, which it does.

This "piecewise" analysis is cumbersome and can be tedious. However, it does provide an insight into what is physi-

The Theory of Transmission Lines (Continued)

cally happening and demonstrates that a complex problem can be solved by dividing it into a series of simpler problems. Also, the mathematics are simple if the exponentials, which provide phase information in the classical transmission line equations, are eliminated. One must provide the "bookkeeping" to combine the reflections at the proper time. This is quite straightforward, since a pulse travels with a constant velocity along an ideal or low loss line and the time delay between reflected pulses can be predicted.

The rules to keep in mind are that at any point and instant of time the voltage or the current is the algebraic sum of the waves traveling in the positive X and the negative X directions. For example, two voltage waves of the same polarity and equal amplitudes, traveling in opposite directions, at a given point and time will add together to yield a voltage of twice the amplitude of the individual wave. The same reasoning applies to points of termination and discontinuities on the line. The total voltage or current is the algebraic sum of all of the incident and reflected waves. Polarities must be observed. A positive voltage reflection results in a negative current reflection and vice versa.

Before considering reflections at the source, due to impedance mismatches between the source impedance and the line impedance, the behavior of the ideal line with various loads will be analyzed when it is driven by a step function.

Step Function Response of the Ideal Line for Various Loads

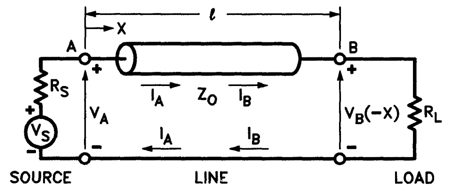
The voltage and current waveforms at point A (line input, Figure 2.3) and point B (the load) for various loads are presented in Table 1. They have been reproduced from Table 5.1, pages 158, 159 of Reference 1. Note that $R_S = Z_O$ and that VA at $t = 0$ is equal to $V_S/2$, which means that there is no impedance mismatch between the source and the line, so there will be no reflection from the source at $t = 2\ T_O$.

T_O is the one way propation of the line.

The time domain response of the reactive loads are obtained by applying a step function to the LaPlace transform of the load and then taking the inverse transform.

Note that the reflection coefficient at the load is not the total reflection coefficient (a complex number) but represents only the real part of the load. The reason for doing this is to eliminate the complex ($j\omega t$) terms because we are performing the bookkeeping involving the phase relationships, which are performed by them in classical transmission line analysis.

Also note that for the open circuit condition, Table 1 (b), ZL = infinity, so that $\rho L = +1$. The voltage is reflected back from the load to the source (at amplitude $V_O = V_S/2$), so that at time $= 2\ T_O$ it adds to the original voltage, $V_O = V_S/2$ to give a value of $2\ V_O = V_S$. During the time the voltage wave is traveling down to and back from the load a current of $I_O = V_O/Z_O = V_S/2\ Z_O$ exists. This current charges up the distributed line capacitance to the value V_S, at which time it stops.



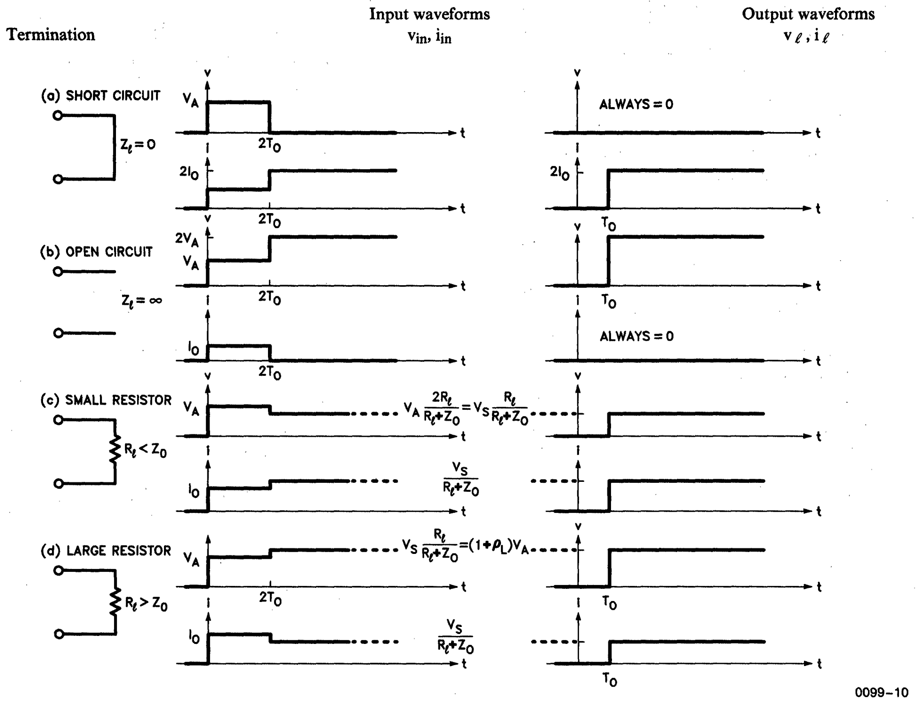
Direction of Travel
VA, IA ⟶ +X
VB, IB ⟵ −X

Figure 2.3. Ideal Transmission Line Loaded and Driven

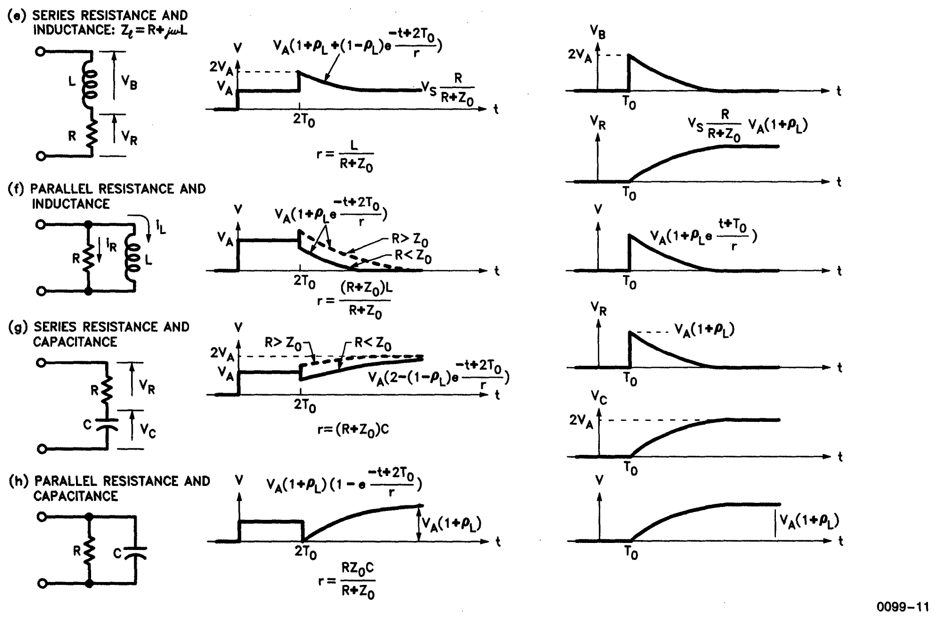
The Theory of Transmission Lines (Continued)

Table 1. Step Function Response of Figure 2.3 for Various Terminations

$V_A = V_S/2,\ I_O = V_O/Z_O,\ T_O = \ell\sqrt{LC},\ \rho L = (R_L - Z_O)/(R_L + Z_O)$



Termination
Input waveforms v_{in}, i_{in}
Output waveforms v_ℓ, i_ℓ
(a) SHORT CIRCUIT
$Z_\ell = 0$
(b) OPEN CIRCUIT
$Z_\ell = \infty$
(c) SMALL RESISTOR
$R_\ell < Z_0$
(d) LARGE RESISTOR
$R_\ell > Z_0$
ALWAYS = 0
$V_A \frac{2R_\ell}{R_\ell + Z_0} = V_S \frac{R_\ell}{R_\ell + Z_0}$
$\frac{V_S}{R_\ell + Z_0}$
$V_S \frac{R_\ell}{R_\ell + Z_0} = (1+\rho_L)V_A$
$2T_0$
T_0
$2I_0$
$2V_A$
0099-10



(e) SERIES RESISTANCE AND INDUCTANCE: $Z_\ell = R + j\omega L$
$V_A(1+\rho_L+(1-\rho_L)e^{\frac{-t+2T_0}{r}})$
$V_S \frac{R}{R+Z_0}$
$r = \frac{L}{R+Z_0}$
$V_S \frac{R}{R+Z_0}\ V_A(1+\rho_L)$
(f) PARALLEL RESISTANCE AND INDUCTANCE
$V_A(1+\rho_L e^{\frac{-t+2T_0}{r}})$
R > Z0
R < Z0
$r = \frac{(R+Z_0)L}{R+Z_0}$
$V_A(1+\rho_L e^{\frac{t+T_0}{r}})$
(g) SERIES RESISTANCE AND CAPACITANCE
$V_A(2-(1-\rho_L)e^{\frac{-t+2T_0}{r}})$
$r = (R+Z_0)C$
$V_A(1+\rho_L)$
(h) PARALLEL RESISTANCE AND CAPACITANCE
$V_A(1+\rho_L)(1-e^{\frac{-t+2T_0}{r}})$
$r = \frac{RZ_0C}{R+Z_0}$
$V_A(1+\rho_L)$
0099-11

The Theory of Transmission Lines (Continued)

The waveforms at the source and load for (g) and (h) are of particular interest because (g) represents a series RC termination that dissipates no DC power and can be used to terminate a transmission line in its characteristic impedance at the input to a Cypress IC. The equivalent circuit of the input to a Cypress IC is represented by (h). The addition of (g) and (h) then models a Cypress IC driven by a transmission line terminated in its characteristic impedance when the values of R and C are properly chosen.

Reflections Due to Discontinuities

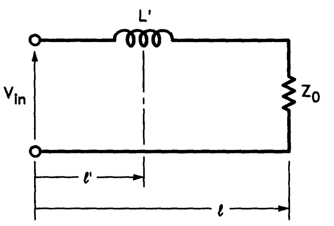
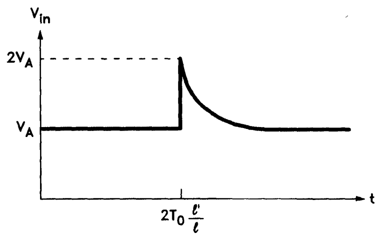
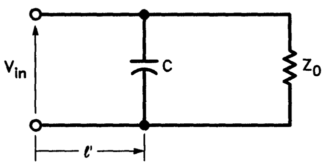
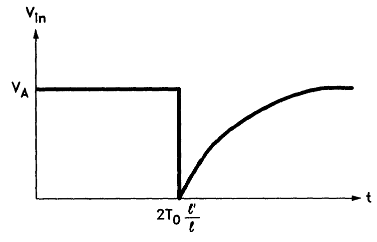
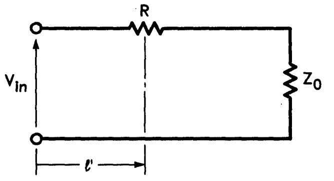
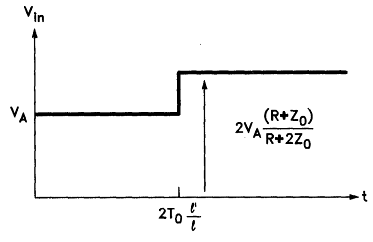
Table 2 illustrates three types of common discontinuities found on transmission lines. When a discontinuity occurs at a point on the line it causes a reflection and some energy is directed back to the source. The amount of energy reflected back is determined by the reflection coefficient at that point. Discontinuities are usually small (by design), so most of the energy is transmitted to the load.

Pulse Response of the Ideal Transmission Line

Consider next the behavior of the ideal transmission line when driven by a pulse whose width is short compared to the electrical length of the line. In other words, when the width of the pulse is less than the one-way propagation delay time, T_O, of the line.

The voltage waveforms at point A (line input, Figure 2.3) and point B (the load) for various loads are presented in Table 3. They have been reproduced from Table 5.2, pages 160, 161 of Reference 1. Note that $R_S = Z_O$ and that VA at $t = 0$ is equal to $V_S/2$, which means that there is no impedance mismatch between the source and the line, so there will be no reflection from the source at $t = 2\ T_O$.

Table 2. Reflections from Discontinuities with an Applied Step Function

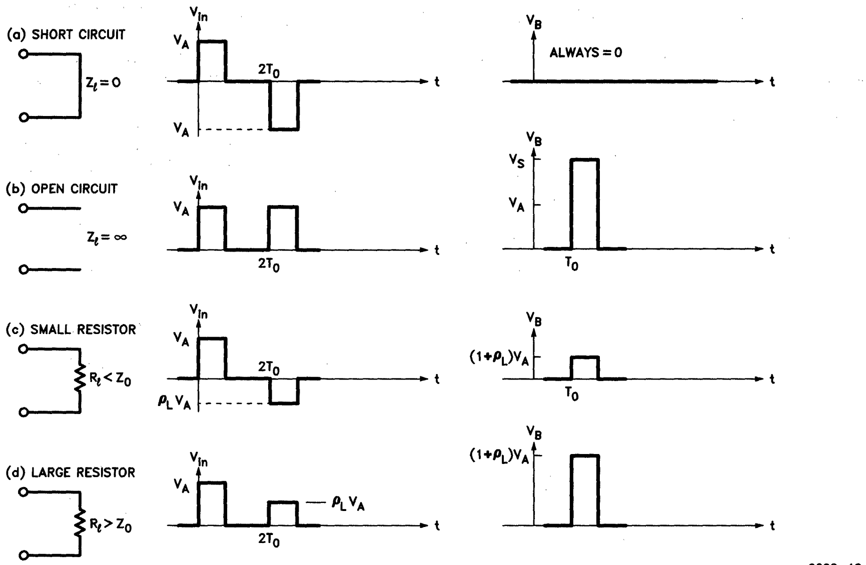
Discontinuity	Voltage Seen at Input End: $V_A = V_S/2$ also, $R_S = Z_O$
(a) Series Inductance  0099-12	 0099-15
(b) Shunt Capacitance  0099-13	 0099-16
(c) Series Resistance  0099-14	 0099-17

Pulse Response of the Ideal Transmission Line (Continued)

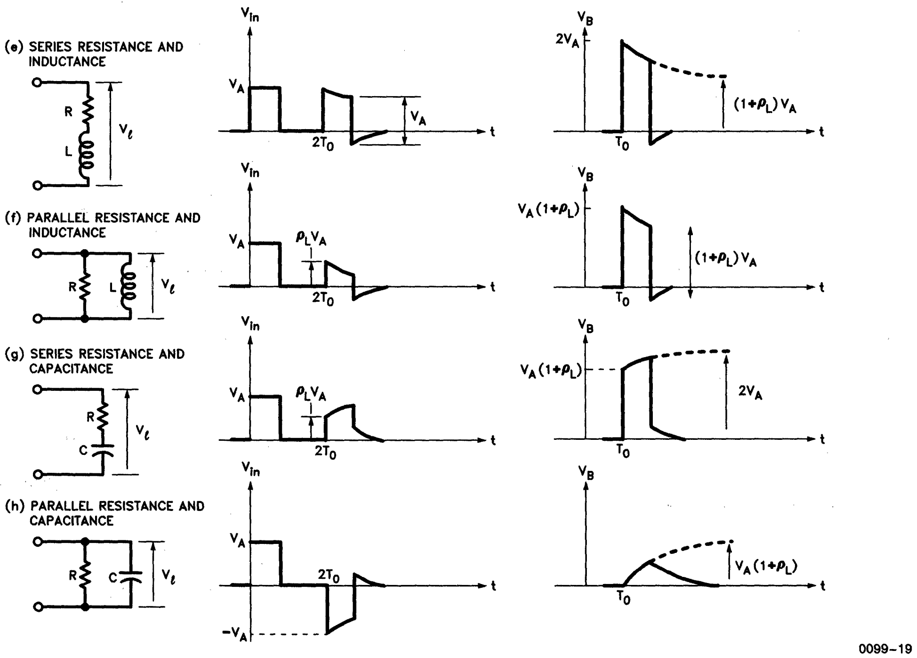
Table 3. Pulse Response of Figure 2-3 for Various Terminations

$V_A = V_S/2,\ T_O = \ell\sqrt{LC},\ \rho L = (R_L - Z_O)/(R_L + Z_O)$

Termination | Input waveform V_{in} | Output waveform v_B



0099-18


(e) SERIES RESISTANCE AND INDUCTANCE
R
L
Vℓ
Vin
VA
2T0
VA
t
VB
2VA
(1+ρL) VA
T0
t
(f) PARALLEL RESISTANCE AND INDUCTANCE
R
L
Vℓ
Vin
VA
ρL VA
2T0
t
VB
VA(1+ρL)
(1+ρL) VA
T0
t
(g) SERIES RESISTANCE AND CAPACITANCE
R
C
Vℓ
Vin
VA
ρL VA
2T0
t
VB
VA(1+ρL)
2VA
T0
t
(h) PARALLEL RESISTANCE AND CAPACITANCE
R
C
Vℓ
Vin
VA
2T0
-VA
t
VB
VA(1+ρL)
T0
t

0099-19

Finite Rise Time Effects

Now consider the effects of step functions with finite rise times driving the ideal transmission line.

If TR is sufficiently fast, the voltage at the load will change in discrete steps. The amplitude of the steps is determined by the impedance mismatch and the width of the steps is determined by the two-way propagation delay of the line.

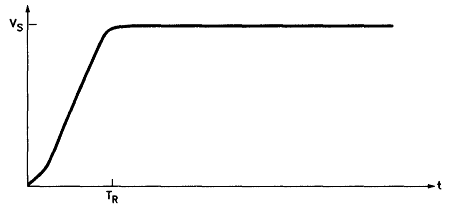
As the risetime becomes slower and the line shorter (smaller T_O), or both, the result converges to the familiar RC time constant, where C is the static capacitance. All devices should be treated as transmission lines for transient analysis when an ideal step function is applied. However, as the rise time becomes larger (slower) and the traces shorter (or both) the transmission line analysis reduces to conventional AC circuit analysis.

Reflections from Small Discontinuities

Table 4 shows a pulse with a linear rise time and rounded edges driving the transmission line of Table 2 (a), (b). The expressions for V_r are derived on pages 171 and 172 of Reference 1. The reflection caused by the small series inductance is useful for calculating the value of the inductor, L′, but little else.

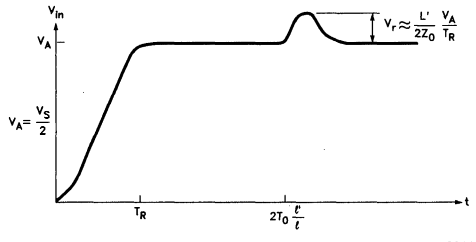
Table 4. Reflections from Small Discontinuities with Finite Rise Time Pulse

(a) Applied Pulse from Generator



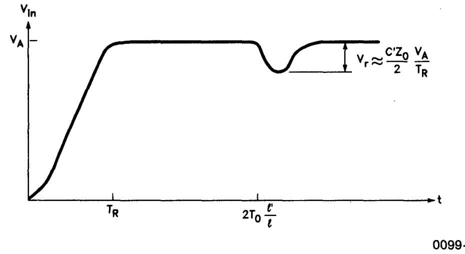
0099–20

(b) Reflection from Small Series Inductor L′



0099–21

(c) Reflection from Small Shunt Capacitor C′

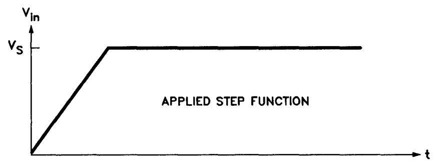


0099–22

The reflection caused by the small shunt capacitor is more interesting because if it is sufficiently large it could cause a device connected to the transmission line to see a logic ZERO instead of a logic ONE.

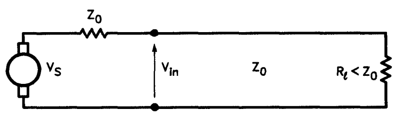
The Effect of Rise Time on Waveforms

Next, consider the ideal line terminated in a resistance less than its characteristic impedance and driven by a step function with a linear rise time. The stimulus, the circuit, and the response are illustrated in *Figures 4.1 (a), (b)* and *(c)*, respectively. Once again, note that the source resistance is equal to the line characteristic impedance, so there are no reflections from the source.



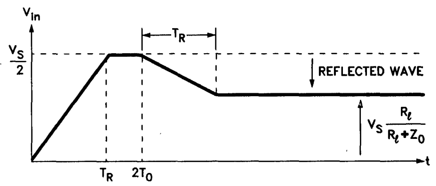
0099–23

(a)



0099–24

(b)



0099–25

(c)

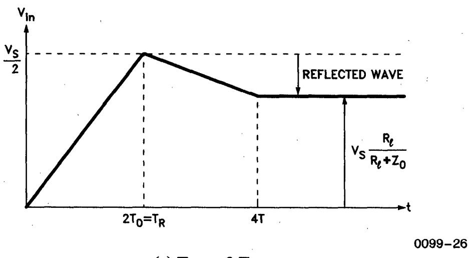
Figure 4.1. Effect of Rise Time on Step Response of Mismatched Line with $R_\ell < Z_O$

The resulting waveforms are similar to those of Table 1 (c) as modified as shown in *Figure 4.1 (c)*. The final value of the waveform must be the same as before (Table 1 (c)).

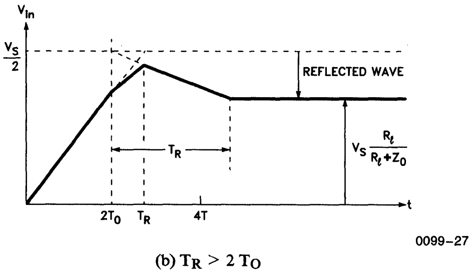
The resultant wave at the line input (V_{in}) is easily obtained by superposition of the applied wave and the reflected wave at the proper time. In *Figure 4.1* the rise time of the step function is less than the (two-way) propagation delay of the line so the input wave reaches its final value, $V_S/2$. At $t = 2\ T_O$ the reflected wave arrives back at the source and subtracts from the applied step function.

The cases where the step function rise time is equal to twice the propagation delay and greater than the propagation delay are illustrated in *Figure 4.2 (a)* and *(b)*, respectively.

Finite Rise Time Effects (Continued)



(a) $T_R = 2\ T_O$



(b) $T_R > 2\ T_O$

Figure 4.2. Effects of Rise Time on Step Response for $R_\ell < Z_O$: (a) $T_R = 2\ T_O$; (b) $T_R > 2\ T_O$

Multiple Reflections and Effective Time Constant

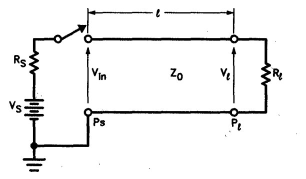
We will now consider the case of an ideal transmission line with multiple reflections causes by improper terminations at both ends of the line. The circuit and waveforms are illustrated in *Figure 4.3.* The reflection coefficients at the source and the load are both negative. i.e., the source resistance and the load resistance are both less than the line characteristic impedance. Refer to equations 2-11 and 2-12.

When the switch is initially closed, a step function of amplitude $V_O = V_{in} = \dfrac{V_S Z_O}{R_S + Z_O}$ appears on the line and travels toward the load. A one-way propagation delay time later, T_O, the wave is reflected back with an amplitude of $\rho L\ V_O$.

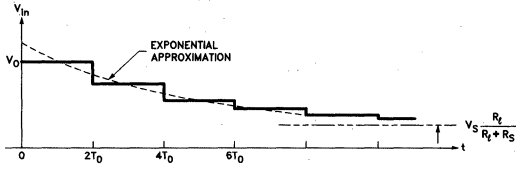
This first reflected wave then travels back to the source and at time $t = 2\ T_O$ it reaches the input end of the line. At this time the first reflection at the source occurs and a wave of amplitude $\rho S\ (\rho L\ V_O)$ is reflected back to the load. At time $t = 3\ T_O$ this wave is again reflected from the load back to the source with amplitude $\rho L\ \rho S\ (\rho L\ V_O) = \rho S\ \rho L^2\ V_O$. This back and forth reflection process continues until the amplitudes of the reflections become so small that they cannot be observed, at which time the circuit is said to be in a quiescent state.

Effective Time Constant

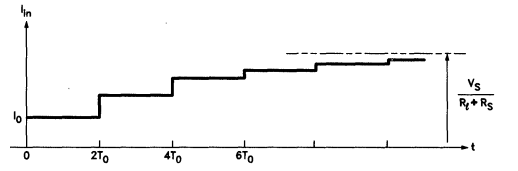
From an examination of *Figure 4.3* it is reasonable that if the voltage reflections occur in small increments that are of short durations the resultant waveform will approximate an exponential function, as indicated by the dashed line in *Figure 4.3 (b).* The smaller and narrower the steps become, the more closely the waveform will approach an exponential.



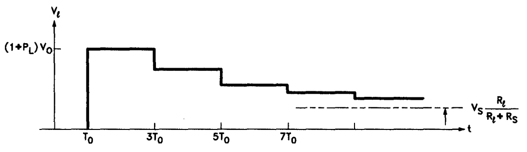
(a)



(b)



(c)



(d)

Figure 4.3. Step Function Applied to Line Mismatched on both ends; waveforms shown for negative values of ρ_S and ρ_ℓ.

The mathematical derivation is presented on pages 178 and 179 of Reference 1. The time constant is shown to be:

$$K = -\frac{2\ T_O}{1 - \rho S\ \rho L} \qquad (4\text{-}1)$$

So that the resultant waveform can be approximated by;

$$V(t) = V_O\ \epsilon \left(\frac{t}{K}\right) \qquad (4\text{-}2)$$

In order for equation 4-2 to be accurate ρL and ρs must be reasonably large (approaching ± 1) so that the incremental steps are small. The product $\rho S\ \rho L$ is a positive number, less than one, so the time constant is a negative number, which indicates that the exponential decreases with time. This is usually the case in transient circuits.

Both reflection coefficients must also have the same sign in order to yield a continually decreasing (or increasing) waveform. Opposite signs will give oscillatory behavior that cannot be represented by an exponential function.

Finite Rise Time Effects (Continued)

The Transition from Transmission Line to Circuit Analysis

When a transmission line is terminated in its characteristic impedance it behaves like a resistor and it usually does not matter if transmission line or circuit analysis is used; provided that the propagation delays are taken into account.

Consider the case of a short-circuited transmission line driven by a step function with a source impedance unequal to the characteristic line impedance. The general case is shown in Figure 4.3 (a). For RL = 0 the reflection coefficients are;

$$\rho S = \frac{Z_S - Z_O}{Z_S + Z_O} \quad \rho L = -1.$$

The approximate time constant is;

$$-k = \frac{2\,T_O}{1 - \rho S\,\rho L} = \frac{2\,T_O}{1 + \rho S} = \frac{T_O\,(Z_S + Z_O)}{Z_S}, \text{ or}$$

$$-k = T_O + \frac{T_O\,Z_O}{Z_S} \qquad (4\text{-}3)$$

Recall that $T_O = \ell\,\sqrt{LC}$ (one-way delay)

and $Z_O = \sqrt{\frac{L}{C}}$, where ℓ is the physical length of the line and L and C are the per-unit-length parameters.

Substitution of these into equation 4-3 yields

$$-k = T_O + \ell\,\frac{L}{Z_S}.$$

It is necessary to have Z_S smaller than Z_O.

Thus the reflection coefficients have the same sign in order to give exponential behavior. Opposite signs give oscillatory behavior.

If $Z_S \ll Z_O$, the exponential approximation becomes more accurate. If Z_S is very small compared to Z_O, then T_O is negligible compared to ℓ L/Z_O, so that equation 4-5 reduces to;

$$k = -\ell\,\frac{L}{Z_S}.$$

But ℓ L is the total loop inductance and Z_S is the total series impedance of the circuit. The time constant is then;

$$k = \frac{L'}{R_S}.$$

This is the same time constant that would have been obtained by a circuit analysis approach if the line were considered a series combination of L′ and R_S.

By open-circuiting the line and performing a similar analysis it can be shown that a RC time constant results.

Types of Transmission Lines

The types of transmission lines are:
- Coaxial cable
- Twisted pair
- Wire over ground
- Microstrip lines
- Strip lines

Coaxial Cable

Coaxial cable offers many advantages for distributing high frequency signals. The well defined and uniform characteristic impedance permits easy matching. The ground shield on the cable reduces crosstalk and the low attenuation at high frequencies make it ideal for transmitting the fast rise and fall time signals generated by Cypress CMOS integrated circuits. However, because of its high cost, coaxial cable is usually restricted to applications where there are no other alternatives. These are usually clock distribution lines on PCBs or backplanes.

Characteristic Impedance

Coaxial cables have characteristic impedances of 50, 75, 93, or 150 ohms. Special cables can be made with other impedances, but these are the most common.

Propagation Delay

The propagation delay is very low. It may be computed using the formula;

$$T_{pd} = 1.017\,\sqrt{e_r}\text{ ns/ft.} \qquad (5\text{-}1)$$

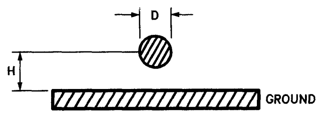
where e_r is the relative dielectric constant and depends upon the dielectric material used. For solid teflon and polyethylene it is 2.3. The propagation delay is 1.54 ns per foot. For maximum propagation velocity, coaxial cables with dielectric styrofoam or polystyrene beads in air may be used. Many of these cables have high characteristic impedances and are slowed considerably when capacitively loaded.

Twisted Pair

Twisted pairs can be made from standard wire (AWG 24-28) twisted about 30 turns per foot. Typical characteristic impedance is 110Ω. Because the propagation delay is directly proportional to the characteristic impedance (equation 2-5) the propagation delay will be approximately twice that of coaxial cable. Twisted pairs are used for backplane wiring and for breadboarding.

Wire Over Ground

Figure 5.1 shows a wire over ground. The wire over ground is used for breadboarding and for backplane wiring. The characteristic impedance is approximately 120Ω and may vary as much as ±40%, depending upon the distance from the groundplane, the proximity of other wires, and the configuration of the ground.



0099-32

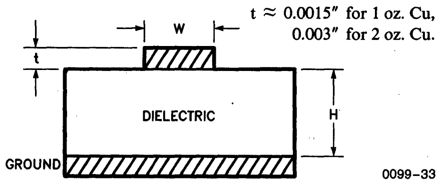
$$Z_O = \frac{60}{\sqrt{e_r}}\ln\left(\frac{4h}{d}\right),$$

Figure 5.1. Wire Over Ground

Types of Transmission Lines (Continued)

Microstrip Lines

A microstrip line *(Figure 5.2)* is a strip conductor (signal line) on a PCB separated from a ground plane by a dielectric. If the thickness and width of the line, and the distance from the ground plane are controlled, the characteristic impedance of the line can be predicted with a tolerance of ±5%.



$$Z_O = \frac{87}{\sqrt{e_r + 1.41}} \ln\left(\frac{5.98H}{0.8w + t}\right),$$

where:

e_r = relative dielectric constant of the board material (about 5 for G-10 fiber-glass epoxy boards),

w, h, t, = dimensions indicated.

Figure 5.2. Microstrip Line

The formula of *Figure 5.2* has proven to be very accurate for ratios of width to height between 0.1 and 3.0 and for dielectric constants between 1 and 15.

The inductance per foot for microstrip lines is;

$$L = Z_O^2 C_O \qquad (5\text{-}2)$$

where Z_O = characteristic impedance,

C_O = capacitance per foot.

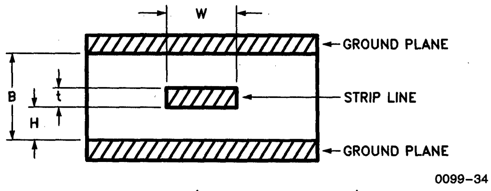
The propagation delay of a microstrip line is;

$$T_{pd} = 1.017 \sqrt{0.45\, e_r + 0.67} \text{ ns per foot.} \qquad (5\text{-}3)$$

Note that the propagation delay is dependent only upon the dielectric constant and is not a function of the line width or spacing. For G-10 fiber-glass epoxy PCBs (dielectric constant of 5), the propagation delay is 1.74 ns per foot.

Strip Line

A strip line consists of a copper strip centered in a dielectric between two conducting planes *(Figure 5.3)*. If the thickness and width of the line, the dielectric constant, and the distance between ground planes are all controlled, the tolerance of the characteristic impedance will be within ±5%. The equation of *Figure 5.3* is accurate for W/(b-t) < 0.35 and t/b < 0.25.



$$Z_O = \frac{60}{\sqrt{e_r}} \ln\left(\frac{4b}{0.67\pi w\left(0.8 + \frac{t}{w}\right)}\right)$$

Figure 5.3. Stripline

The inductance per foot is given by the formula;

$$L_O = Z_O^2 C_O.$$

The propagation delay of the line is given by the formula;

$$T_{pd} = 1.017 \sqrt{e_r} \text{ ns per foot.} \qquad (5\text{-}4)$$

For G-10 fiber-glass epoxy boards the propagation delay is 2.27 ns per foot. The propagation delay is not a function of line width or spacing.

Power Distribution

Instantaneous Current

In order to realize the fast rise and fall times that Cypress CMOS integrated circuits are capable of achieving, the power distribution system must be capable of supplying the instantaneous current required when the device outputs switch from LOW to HIGH.

The energy is stored as charge on the local decoupling capacitors. It is standard practice to use one decoupling capacitor for each IC that drives a transmission line and to use one for every three devices that do not.

The value of the decoupling capacitor is determined by estimating the instantaneous current required when all the outputs of the IC switch from LOW to HIGH, assuming a reasonable "droop" of the voltage on the capacitor.

Calculations

The charge stored on the local decoupling capacitor of *Figure 6.1* is Q = C V. Differentiating yields;

$$i(t) = \frac{dQ}{dt} = C\frac{dV}{dt}. \qquad (6\text{-}1)$$

The characteristic impedance of a typical transmission line is 50Ω. Heavily (capacitively) loaded lines will have lower characteristic impedances (equation 2-7).

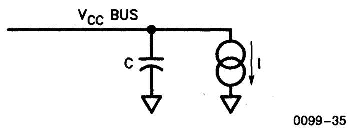


Figure 6.1. Local Decoupling Capacitor

Next, assume that the IC is an eight output PROM, such as the CY7C245 or the CY7C261. The outputs will reach V_{CC} -Vt = 5V-1V = 4V. Each output will then require 4V/50 = 8 mA. Since there are eight outputs a total of 64 mA will be required.

Solving equation 6-1 for C yields;

$$C = I\frac{dt}{dV}. \qquad (6\text{-}2)$$

The signal rise and fall times are 2 to 4 ns so we will use dt = 3 ns.

The last step is to assume a reasonable, tolerable droop in the capacitor voltage. Assume dV = 100 mV.

Therefore, substituting these values in equation 6-2 yields;

$$C = \frac{64 \times 10^{-3} \times 3 \times 10^{-9}}{100 \times 10^{-3}} = 0.192 \times 10^{-9} = 192\,\text{pF}.$$

It is standard practice to use 0.01 to 0.1 μF decoupling capacitors. A 0.01 μF capacitor is capable of supplying 330 mA under the preceding conditions.

Power Distribution (Continued)

Decoupling capacitors for high speed Cypress CMOS circuits should be of the high K ceramic type with a low ESR (Equivalent Series Resistance). Capacitors using 5 ZU dielectric are a good choice.

Low Frequency Filter Capacitors

A solid tantalum capacitor of 10 μF is recommended for each 50 to 100 ICs to reduce power supply ripple. This capacitor should be as close as possible to where the V_{CC} and ground enter the PCB or module.

When Should Transmission Lines Be Terminated?

Transmission lines should be terminated when they are long. From the preceding analysis it should be apparent that

$$\text{Long Line} > \frac{T_r}{2\,T_{pd}}.$$

Where T_{pd} is the propagation delay per unit length.

For Cypress products, the rise time, T_r, is typically two nanoseconds.

The propagation delay per unit length has been shown to be as small as 1.7 ns per foot.

$$\text{Long Line} > \frac{2\text{ ns}}{2 \times 1.7\text{ ns/ft.}} = 0.59\text{ ft. or 7 inches.}$$

Not all lines exceeding 7 inches will need to be terminated. Terminations are usually only required on clock inputs, write and read strobe lines on SRAMs, and chip select or output enable lines on RAMs, PROMs, and PLDs. Address lines and data lines on RAMs and PROMs usually have time to settle.

In the case where multiple loads are connected to a transmission line, only one termination circuit is required. The termination network should be located at the load that is electrically the longest distance from the source. This is usually the load that is the longest physical distance from the source.

Types of Terminations

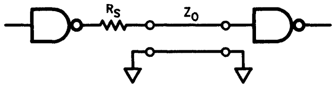
There are three basic types of terminations. They are called series damping, parallel, and pullup/pulldown. Each has their advantages and disadvantages.

Except for series damping, the termination network should be attached to the input (load) that is electrically furthest away from the source. Component leads should be as short as possible in order to prevent reflections due to lead inductance.

Series Damping

Series damping is accomplished by inserting a small resistor (typically 10Ω to 75Ω) in series with the transmission line, as close to the source as possible, as illustrated in *Figure 8.1*. Series damping is a special case of damping in which the series resistor value plus the circuit output impedance is equal to the transmission line impedance. The strategy is to prevent the wave that is reflected back from the load from reflecting back from the source by making the source reflection coefficient equal to zero.

The channel resistance (ON resistance) of the pulldown device for Cypress ICs is ten to twenty ohms (depending upon the current sinking requirements), so this value should be subtracted from the series damping resistor, R_S.



0099-36

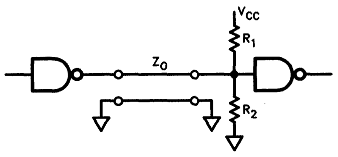
Figure 8.1. Series Damping

The disadvantage of the series damping technique is that during the two-way propagation delay time the voltage at the input to the line is half-way between the logic levels, due to the voltage divider action of R_S. This means that no inputs can be attached along the line, because they would respond incorrectly. However, any number of devices may be attached to the load end of the line because all of the reflections will be absorbed at the source.

Due to the low input current required by Cypress CMOS ICs, there will be essentially no DC power dissipation and the only AC power required will be to charge and discharge the parasitic capacitances.

Pullup/Pulldown

The pullup/pulldown resistor termination shown in *Figure 8.2* is included only for the sake of completeness. If both resistors are used there will be DC power dissipated all the time and if only a pulldown resistor is used DC power will be dissipated when the input is in the logic HIGH state. Due to these power dissipations, this termination is not recommended.



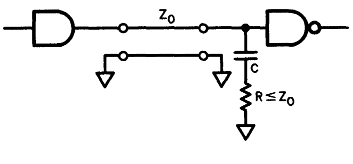
0099-37

Figure 8.2. Pullup/Pulldown

However, in special cases where inputs should be either pulled up (HIGH) for logic reasons or because of very slow rise and fall times, a pullup resistor to V_{CC} may be used in conjunction with the terminating network described below. DC power will be dissipated when the source is LOW.

Parallel AC Termination; *Figure 8.3*

This is the recommended general purpose termination. It does not have the disavantage of the half-voltage levels of series damping and it causes no DC power dissipation. Loads may be attached anywhere along the line and they will see a full voltage swing.



0099-38

Figure 8.3. Parallel AC

Types of Terminations (Continued)

The disadvantage is that it requires two components, versus the series damping termination of one. The value of the terminating resistor, R, should be slightly less than the line characteristic impedance.

Low Pass Filter Analysis

The parallel AC termination has a second advantage: it acts as a low-pass filter for short pulses.

This can be verified by analysis of the response of the circuit, illustrated in *Figure 8.4,* to a positive and to a negative step function. The positive step function is generated by moving the switch from position 2 to position 1. The negative step function is generated by moving the switch from position 1 to position 2. The response of the circuit to a pulse is then the superposition of the two responses. The input impedance of the Cypress circuits that are connected to the termination network are so large that they may be ignored for this analysis.

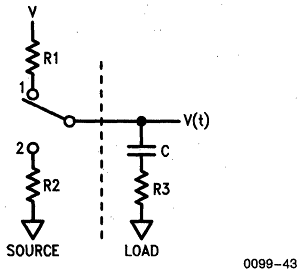


Figure 8.4. Lumped Load

Classic circuit analysis usually assumes an ideal (R1 = R2 = 0) source. In real-world digital circuits the source output impedance is not only non-zero, but also different depending upon whether the output is changing from LOW to HIGH or vice versa.

For Cypress integrated circuits, $100\Omega > R1 > 30\Omega$ and $20\Omega > R2 > 10\Omega$, depending upon speed and output current sinking specifications.

Positive Step Function Response

The initial voltage on the capacitor is zero. At t = 0 the switch is moved from position 2 to position 1. At t = 0+ the capacitor appears as a short circuit and the voltage V is applied through R1 to charge the load (R3 C). The voltage between the capacitor and ground, V(t), is;

$$V(t) = V\left(1 - \epsilon^{\frac{-t}{(R1 + R3)C}}\right) \qquad (8\text{-}1)$$

In theory, the voltage across the capacitor reaches V when t equals infinity.

In practice, the voltage reaches 98% of V after 3.9 RC time constants. This can be verified by setting V(t)/V = 0.98 in equation 8-1 and solving for t.

Negative Step Function Response

The capacitor is charged to (approximately) V. At t = 0 the switch is moved from position 1 to position 2 and the capacitor is discharged. The voltage between the capacitor and ground, V(t), is;

$$V(t) = V\epsilon^{\frac{-t}{(R2 + R3)C}} \qquad (8\text{-}2)$$

The voltage decays to 2% of its original value in 3.9 RC time constants. This can be verified by setting V(t)/V = 0.02 in equation 8-2 and solving for t.

First, the Ideal Case

Consider first the ideal case where R1 = R2 = 0. Let R3 = R in equations 8-1 and 8-2. If a positive pulse of width T is applied to the circuit of *Figure 8.4,* it will disappear if 4RC > T.

Because the discharge time constant is the same as the charging time constant for the ideal case, a negative going pulse of width T will also disappear if 4RC > T. i.e., if the applied signal were normally HIGH and went LOW, such as a write strobe on a SRAM, all negative glitches will be filtered out if they are less than 4RC time constants in width.

The maximum frequency that the circuit will pass is;

$$F\,(\text{max.}) = \frac{1}{2\,T}. \qquad (8\text{-}3)$$

This is true because the charging and discharging time constants are equal for the ideal case.

Determination of the Capacitance, C, for the Ideal Case

The value of the capacitor, C, must be chosen to satisfy two conflicting requirements. First, it should be large enough to either absorb or supply the energy contained or removed when positive-going or negative-going glitches occur. Second, it should be small enough not to either delay the signal beyond some design limit or to slow the signal rise and fall times to greater (i.e., longer timewise) than 5 ns.

A third consideration is the impedance caused by the capacitive reactance, XC, of the capacitor. The digital waveforms applied to the AC termination can be expressed in terms of Fourier Series so that they can be manipulated mathematically. However, because these digital signals are not "periodic" in the classical meaning of the word, it is not clear that the "AC steady state analysis model" of XC is applicable.

In most applications, the degradation of the signal rise and fall times beyond 5 ns determines the maximum value of the capacitor. The procedure will be to calculate the risetime between the 10% and 90% amplitude levels, equate this to 5 ns, and solve for C in terms of R.

Types of Terminations (Continued)

Solving the equation $V(t) = V(1 - \epsilon^{\frac{-t}{RC}})$ for t yields;

$$t = R\,C \ln \left[\frac{1}{1 - \frac{V(t)}{V}} \right] \qquad (8\text{-}4)$$

For $\frac{V(t)}{V} = 0.1$, $t = 0.10\,R\,C$.

For $\frac{V(t)}{V} = 0.9$, $t = 2.3\,R\,C$.

The time for the signal to transition from 10% to 90% of its final value is then T = 2.2 R C. Solving for C yields;

$$C = \frac{T}{2.2\,R} \qquad (8\text{-}5)$$

For T = 5 ns the following table may be constructed.

	PCB	Wirewrap
Z_O (Ω)	50	120
R (Ω)	47	110
C (max., pF)	48	20
RC (ns)	2.25	2.2
4RC (ns)	9	8.8

What this table says is that 50Ω transmission lines on printed circuit boards that are terminated with RC networks should use a 47Ω resistor and a maximum capacitor of 48 pF. Under this condition, glitches of 9 ns or less will be eliminated. The second column applies to wirewrap construction.

Then for the Real World

The value of R1 and R2 should be determined from the data sheet.

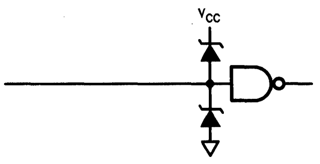
The value of R1 should be added to 47Ω and C then calculated using equation 8-5. Next, check to see that the charging RC time constant does not violate some minimum positive pulse width specification for the particular line. If so, reduce C.

Add the value of R2 to 47Ω and calculate C. Then check if the discharging RC time constant violates some minimum negative pulse width specification for the particular line. If so, reduce C.

Schottky Diode Termination

In certain instances it may be expedient to use Schottky diodes to terminate lines. Where line impedances are not well defined, as in breadboards and backplanes, the use of diode terminations is convenient and may save time.

A typical diode termination is shown in *Figure 9.1.* The low forward voltage, V_f, of the diode (typically 0.3 to 0.45V) clamps the input signal to a V_f below ground (lower diode) and $V_{CC}+V_f$ (upper diode), thereby significantly reducing signal undershoot and overshoot. In some applications both diodes may not be required.



0099-39

Figure 9.1. Schottky Diode Termination

The advantages of diode terminations are:

- Impedance matched lines are not required.
- The diodes replace terminating resistors or RC terminations.
- The clamping actions of the diodes reduce overshoot and undershoot.
- Although diodes are more expensive than resistors, the total cost of layout may be less because a precise, controlled transmission line environment is not required.
- If ringing is discovered to be a problem during system checkout the diodes can be easily added.

As with resistor or RC terminations, the leads should be as short as possible in order to avoid ringing due to lead inductance.

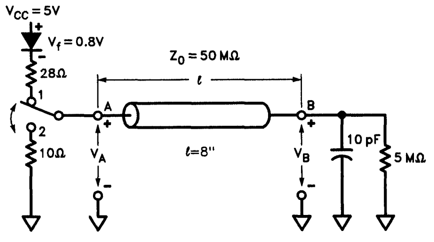
A few of the types of Schottky diodes commercially available are :

- 1N4148 (Switching)
- 1N5711
- MBD101 (Motorola)
- HP5042 (Hewlett Packard)

Example: Unterminated Line

The following example is presented to illustrate the procedure for calculating the waveforms when a Cypress PLD is used to generate the write strobe for a Cypress SRAM. The PLD is a PAL®C 20 device and the SRAM is the CY7C189-25.

The equivalent circuit is illustrated in *Figure 10.1* and the (unmodified) driving waveform in *Figure 10.2.* The rise and fall times are two nanoseconds. The length of the microstrip trace on the PCB is eight inches and the characteristic line impedance is 50Ω. It is required to calculate the voltage waveforms at the source (point A) and the load (point B) as functions of time.



0099-40

Figure 10.1. Equivalent Circuit for Cypress PAL Driving RAM

Example: Unterminated Line (Continued)

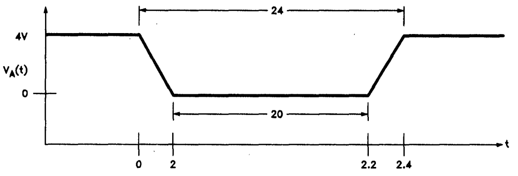


Figure 10.2. V_A (t), Unmodified

Equivalent Circuits for The PLD and SRAM

The equivalent ON channel resistance of the PLD pullup device, 28Ω, was calculated using the output source current versus voltage graph over the region of interest (0 to 2V) from the data sheet. The equivalent resistance of the pulldown device, 10Ω, was calculated in a similar manner, using the output sink current versus output voltage graph, also on the data sheet.

The equivalent input circuit for the SRAM was constructed by approximating the input and stray capacitance with a 10 pF capacitor and the resistance with a 5 million ohm resistor. The input leakage current for all Cypress products is specified as a maximum of ± 10 μA, which guarantees a minimum of 500,000Ω at V_{in} = 5V. Typical leakage current is one microampere.

Transmission Line Calculations

The next step is to calculate the propagation delay and loaded characteristic impedance of the line.

Propagation Delay

The unloaded propagation delay of the line is calculated using equation 5-3 with a dielectric constant of 5.

$$T_{pd} = 1.74 \text{ ns/ft.}$$

In order to calculate the loaded line propagation delay, the intrinsic capacitance must first be calculated using equation 2-5.

$$T_{pd} = Z_O C_O,$$

where Z_O is the intrinsic characteristic impedance and C_O is the intrinsic capacitance.

$$C_O = \frac{T_{pd}}{Z_O} = \frac{1.74 \text{ ns/ft.}}{50} = 34.8 \text{ pF/ft.}$$

The line is loaded with 10 pF, so equation 2-6 is used to compute the loaded propagation delay of the line.

$$T_{pd}' = T_{pd}\sqrt{1 + \frac{C_D}{C_O}}$$

$$T_{pd}' = 1.74 \text{ ns/ft.} \sqrt{1 + \frac{10 \text{ pF}}{34.8 \text{ pF/ft.} \times \frac{8 \text{ in.}}{12 \text{ in./ft.}}}}$$

$$T_{pd}' = 2.08 \text{ ns/ft.}$$

Note that the capacitance per unit length must be multiplied by the line length to arrive at an equivalent lumped capacitance.

Characteristic Impedance

The intrinsic line impedance is reduced by the same factor by which the propagation delay is increased (1.96). See equation 2-7.

$$Z_O' = \frac{50\Omega}{1.196} = 41.8\Omega.$$

Initial Conditions

At time t = 0 the circuit of *Figure 10.1* is in a quiescient state. The voltage at points A and B must be the same.

By inspection;

$$VA = VB = (V_{CC} - Vf)\left(\frac{RL}{RS + RL}\right)$$

$$= (5\text{-}1)\left(\frac{5 \times 10^6}{28 + 5 \times 10^6}\right) = 4V$$

The Falling Edge of the Write Strobe

At t = 0 the driving waveform changes from 4V to 0V (approximately) with a fall time of two nanoseconds. This is represented in *Figure 10.1* by the switch arm moving from position 1 to position 2. The wave propagates to the load at the rate of 2 ns per foot (approximately) and arrives there

$$T_O = 2 \text{ ns/ft.} \times \frac{8 \text{ in.}}{12 \text{ in./ft.}} = 1.33 \text{ ns}$$

later, as illustrated in *Figure 10.3 (b).*

The reflection coefficient at the load is ρL = 1, so a nearly equal and opposite polarity waveform is propagated back to the source from the load, arriving at t = 2 T_O = 2.66 ns, as shown in *Figure 10.3 (a).* (See Table 3 {h}). Note that the falltime is preserved. The reflection coefficient at the source is;

$$\rho s = \frac{RS - Z_O'}{RS + Z_O'} = \frac{10 - 41.8}{10 + 41.8} = -0.61$$

The magnitude of the reflected voltage at the source is then;

$$VS1 = -4V \times (-0.61) = 2.44V.$$

This wave propagates from the source to the load and arrives at t = 3 T_O, and adds to the (zero volts) signal. The risetime is preserved, so the time required for the signal to go from 0V to 2.44V is;

$$t_r = 2.44V \times 2 \text{ ns/4V} = 1.22 \text{ ns.}$$

The signal at the load thus reaches the 2.44V level at time t = 3 T_O + 1.22 ns = 5.22 ns and remains at that level until the next reflection occurs at t = 5 T_O. The wave that arrived at the load at 3 T_O is reflected back to the source and arrives at t = 4 T_O (5.32 ns). The 2.44V level adds to the −4V level, so that the resultant level is −1.56V. The risetime is preserved, so that this level is reached at t = 4 T_O + 1.22 ns = 6.54 ns, and maintained until the next reflection occurs at t = 6 T_O. The 2.44V wave that arrived at the source at t = 4 T_O is reflected back to the load and arrives at t = 5 T_O. The portion that is reflected back is;

$$VS2 = 2.44 \times (-0.61) = -1.49V.$$

Example: Unterminated Line (Continued)

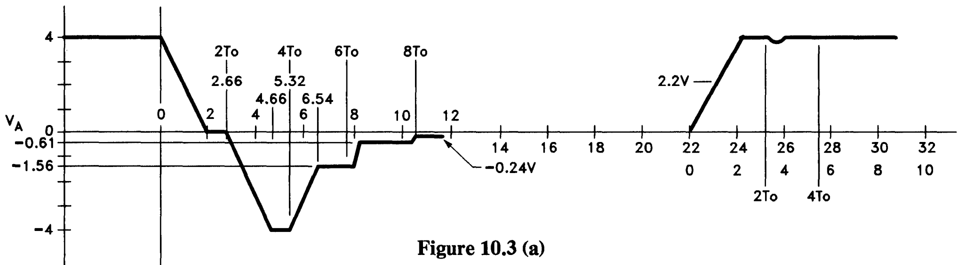


Figure 10.3 (a)

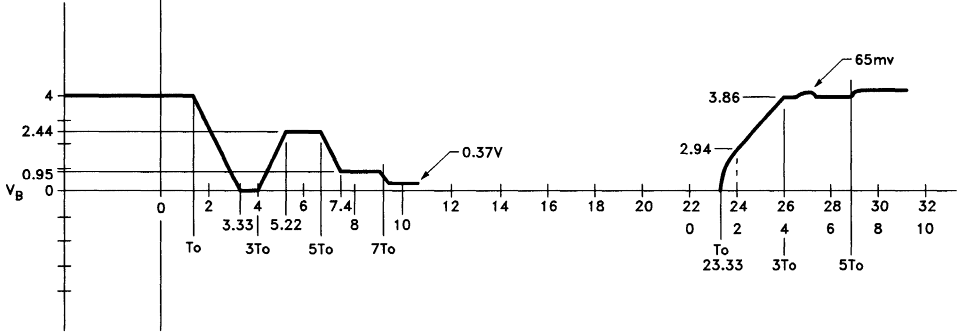


Figure 10.3 (b)

This subtracts from the 2.44V level to give 2.44 −1.49 = 0.95V. The falltime is preserved, so the time required for the signal to go from 2.44V to 0.95V is;

$$t_f = 1.49V \times 2\ ns/4V = 0.75\ ns.$$

The 0.95V level is thus reached at time t = 5 T_O + 0.75 ns = 7.4 ns.

At t = 6 T_O the 0.95V wave arrives back at the source, where it subtracts from the −1.56V level to give −0.61V. The risetime is t_r = 0.95 × 0.5 ns/V = 0.45 ns.

The 0.95V wave that arrived at the source at t = 6 T_O is reflected back to the load and arrives at t = 7 T_O. The portion that is reflected back is;

$$VS3 = 0.95 \times (-0.61) = -0.58V.$$

This subtracts from the 0.95V level to give 0.37V. The falltime is approximately 0.5 ns.

This process continues until the voltages at points A and B decay to approximately zero volts.

Observations

The positive reflection coefficient at the load and the negative reflection coefficient at the source result in an oscillatory behavior that eventually decays to acceptable levels. The voltage at point A reaches −0.61V after 6 T_O delays and the voltage at point B reaches 0.37V after 7 T_O delays.

The reflection at the load that causes the voltage to exceed the TTL minimum ONE level (2V) at T = 3 T_O could cause a problem if either the data to be written in the RAM changes up to 5 T_O delays after the falling edge of the write strobe or if the observed shortening of the write strobe by 5 T_O delays violates the minimum write strobe specification.

However, if this reflection occurred on a clock line to a logic device, registered PROM, or a PLD the reflection could be interpreted by the device as a second clock. The width of the pulse caused by the reflection in this case is 2 T_O = 2.66 ns, which is probably too short to be detected. If the line were either slightly longer or more heavily

Example: Unterminated Line (Continued)

capacitively loaded the pulse would be wider and could be detected as a second clock.

The Rising Edge of the Write Strobe

At t = 22 ns the rising edge of the write strobe begins, which is the equivalent of closing the switch in *Figure 10.1* in the 1 position. For this analysis it its convenient to start the time scale over at zero, as is shown in *Figures 10.3 a* and *b.*

If the forcing function were a step function, the equations of Table 1 (h) would apply. The time constant in the equation is:

$$T = \frac{R\,Z_O'\,Ce}{R + Z_O'}. \qquad (10\text{-}1)$$

Because $R \gg Z_O'$, $T = Z_O'\,Ce$, where $Z_O' = 41.8\Omega$ and $Ce = 33.2$ pF.

This is the equivalent of saying that the five megohm device input resistance can be ignored for transient circuit analysis. Substitution of Z_O' and Ce into the preceding equation yields a time constant of T = 1.39 ns.

Writing the equation for the voltages for the circuit of *Figure 10.1*

$$VA(t) = i\,Z_O' + \frac{1}{Ce}\int_o^t i\,dt. \qquad (10\text{-}2)$$

Also, $VA(t) = Kt\,U(t) - K(t - T1)\,U(t - T1)$. (10-3)

Where Kt is the rising edge of the write strobe (K = 2V/ns) applied at t = 0 using a unit step function, U(t), and −K (t − T1) represents an equal but opposite waveform applied at t = T1 (after the risetime) using a unit step function, U(t − T1).

Equating the equations and taking the LaPlace transforms of both sides yields:

$$\frac{K}{s^2} - \frac{K\epsilon^{-T1S}}{s^2} = Z_O'\,I(s) + \frac{I(s)}{Ce\,s} = \left(Z_O' + \frac{1}{Ce\,s}\right) I(s). \qquad (10\text{-}4)$$

$$\text{However, } VB(t) = \frac{1}{Ce}\int_o^t i\,dt, \text{ or } VB(s) = \frac{I(s)}{Ce\,s}.$$

Therefore:

$$\frac{K}{s^2} - \frac{K\epsilon^{-T1S}}{s^2} = \left(Z_O' + \frac{1}{Ce\,S}\right) Ce\,s\,VB(s). \qquad (10\text{-}5)$$

Solving for VB(s) yields:

$$VB(s) = \frac{\frac{K}{s^2}\left(1 - \epsilon^{-T1S}\right)}{Ce\,s\left(Z_O' + \frac{1}{Ce\,s}\right)}. \qquad (10\text{-}6)$$

Which is equivalent to:

$$VB(s) = \frac{\frac{K}{Z_O'\,Ce}\left(1 - \epsilon^{-T1S}\right)}{s^2\left(S + \frac{1}{Z_O'\,Ce}\right)}. \qquad (10\text{-}7)$$

Taking the inverse LaPlace transform yields:

$$VB(t) = \left[K\,Z_O'\,Ce\left(\epsilon^{\frac{-t}{Z_O'\,Ce}} - 1\right) + Kt\right]U(t) - \qquad (10\text{-}8)$$

$$\left[K\,Z_O'\,Ce\left(\epsilon^{\frac{-(t-T1)}{Z_O'\,Ce}} - 1\right) + K(t - T1)\right]U(t - T1)$$

Equation 10-8 consists of two terms. The first term applies from time zero up to and including T1 and the second term applies after T1.

$$VB(t) = \frac{K\,Z_O'\,Ce}{T1}\left(\epsilon^{\frac{-t}{Z_O'\,Ce}} - 1\right) + \frac{K}{T1}(t)\quad t \le T1 \qquad (10\text{-}9)$$

$$VB(t) = \frac{K\,Z_O'\,Ce}{T1}\left(1 - \epsilon^{\frac{T1}{Z_O'\,Ce}}\right)\epsilon^{\frac{-t}{Z_O'\,Ce}} + K1\quad t > T1 \qquad (10\text{-}10)$$

where K1 = final value = 4V

Substitution of the proper values into equation 10-9 yields at t = T1 = 2 ns.

VB (t = T1) =

$$\frac{2 \times 41.8 \times 33.2 \times 10^{-12}}{2 \times 10^{-9}}\left(\epsilon^{-1.439} - 1\right) + \frac{2V}{ns} \times 2\,ns$$

$$= -1.057 + 4 = 2.94V$$

If the forcing function would have been a step function the equation would be:

$$VB(t) = 4V\left(1 - \epsilon^{\frac{-t}{Z_O'\,Ce}}\right) \qquad (10\text{-}11)$$

at t = 2 ns, VB = 3V, which is greater than the 2.94V calculated using equation 10-9.

At t = (22 ns) + T_O the voltage waveform begins to build up at the load and continues to build until the first reflection from the source occurs at t = 3 T_O.

Equation 10-10 is used to calculate the voltage at the load at t = 2 T_O (because 1 T_O is used for propagation delay time).

VB (t = 2 T_O) =

$$\frac{-2V \times 41.8 \times 33.2 \times 10^{-12}}{2 \times 10^{-9}}\left(1 - \epsilon^{-1.439}\right)\left(\epsilon^{-2}\right) + 4$$

$$= -1.39\,(0.762)\,(0.135) + 4$$

$$= -0.143 + 4 = 3.86V$$

The voltage at the load will remain at this value until the first reflection from the source reaches the load at t = 3 T_O.

Meanwhile, at t = T_O, the wave at the load is reflected back to the source and arrives there at t = 2 T_O. It subtracts from the 4V level at the source as illustrated in Table 4 (c). The amplitude of the "droop" is given by:

$$V_r \cong \frac{C'\,Z_O'}{2}\,\frac{V_O}{TR} \qquad (10\text{-}11)$$

for the case Vs = Z_O'.

Example: Unterminated Line (Continued)

If Vs $\neq Z_O'$ equation 10-11 must be modified. Instead of $\frac{V_O}{2}$ the voltage is $V_O\left(\frac{R_S}{R_S + Z_O'}\right)$, so that equation 10-11 becomes:

$$V_r \cong \frac{C'\,Z_O'\,V_O}{TR}\left(\frac{R_S}{R_S + Z_O'}\right). \qquad (10\text{-}12)$$

where: $C' = 10$ pF

$Z_O' = 41.8\Omega$

$R_S = 28\Omega$

$T_R = 2$ ns

$V_O = 4V$

Substitution of these values into equation 10-12 yields:

$$V_r = 0.33V.$$

$4V - 0.33V = 3.67V$, so there is no danger of the voltage dropping below the minimum HIGH level.

The reflection coefficient at the source is:

$$\rho s = \frac{R_S - Z_O'}{R_S + Z_O'} \text{ where: } \begin{array}{l} R_S = 28\Omega \\ Z_O' = 41.8\Omega \end{array}$$

$$\rho s = -0.198$$

The amount of voltage reflected from the source back to the load is then:

$$VS_1 = (-0.33) \times (-0.198) = +0.065V.$$

This same result could have been obtained by applying the ramp function of *Figure 10.2* to a large resistor and then to a capacitive load and adding the results using superposition.

Observations

The risetime of the waveform at the load is reduced by the 10 pF load capacitor. The reflection at the source caused by the load capacitor is insufficient to reduce the 4V level to less than the TTL one level (2V).

The reflection coefficient at the source is sufficiently small so that the energy reflected back to the load is insufficient to cause a problem.

Summary

The example has demonstrated that, under certain conditions, the voltage reflections caused by the impedance mismatch between a PCB trace and the input of a Cypress CMOS integrated circuit may cause a pulse whose energy is sufficient to be detected by another circuit.

It is the responsibility of the system designer to identify and to analyze these conditions and to then modify the design such that the reflections will not occur.

References

1. Matick, Richard E,: *Transmission Lines For Digital and Communications Networks,* McGraw Hill, 1969.
2. Blood, Jr, William R.,: *MECL System Design Handbook,* Motorola Inc., 1983.

NOTES:

CYPRESS SEMICONDUCTOR

Power Characteristics of Cypress Products

Introduction

SCOPE AND PURPOSE

This document presents and analyzes the power dissipation characteristics of Cypress products. The purpose of this document is to provide the user with the knowledge and the tools to manage power when using Cypress CMOS products.

DESIGN PHILOSOPHY

The design philosophy for all Cypress products is to achieve superior performance at reasonable power dissipation levels. The CMOS technology, the circuit design techniques, architecture and the topology have been carefully combined in order to optimize the speed/power ratio.

SOURCES OF POWER DISSIPATION

Power is dissipated within the integrated circuit as well as external to it. Both internal and external power have a quiescent (or DC) component and a frequency dependent component. The relative magnitudes of each depend upon the circuit design objectives. In circuits designed to minimize power dissipation at low to moderate performance, the internal frequency dependent component is significantly greater than the DC component. In the high performance circuits designed and manufactured by Cypress, the internal frequency dependent power component is much less than the DC component. The reason for this is that a large percentage of the internal power is dissipated in linear circuits such as sense amplifiers, bias generators and voltage/current references that are required for high performance.

External Power Dissipation

The input impedance of CMOS circuits is extremely high. As a result, the DC input current is essentially zero (10 μA or less). When CMOS circuits drive other CMOS circuits there is practically no DC output current. However, when CMOS circuits drive either bipolar circuits or DC loads, external DC power is dissipated. It is standard practice in the semiconductor industry to NOT include the current from a DC load in the device I_{CC} specification. Cypress supports this practice. It is also standard practice to NOT include the current required to charge and discharge capacitive loads in the data sheet I_{CC} specification. Cypress also supports this standard practice.

Frequency Dependent Power

CMOS integrated circuits inherently dissipate significantly less power than either bipolar or NMOS circuits. In the ideal digital CMOS circuit there is no direct current path between V_{CC} and V_{SS}; in circuits using other technologies such paths exist and DC power is dissipated while the device is in a static state.

The principal component of power dissipation in a power-optimized CMOS circuit is the transient power required to charge and discharge the capacitances associated with the inputs, outputs, and internal nodes. This component is commonly called CV^2f power and is directly proportional to the operating frequency, f. The corresponding current is given by the formula

$$I_{CC}(f) = CVf.$$

The primary sources of frequency dependent power are due to the capacitances associated with the internal nodes and the output pins. For "regular" logic structures, such as RAMs, PROMs and FIFOs the internal capacitances are "balanced" so that the same delay and, therefore, the same frequency dependent power is dissipated independent of the location that is addressed. This is not true for programmable devices such as PALs because the capacitive loading of the internal nodes is a function of the logic implemented by the device. In addition, PALs and other types of logic devices may contain sequential circuits so the input frequency and the output frequency may be different.

The capacitance of each input pin is typically 5 pF, so its contribution to the total power is usually insignificant.

Note:
The Cypress Power/Speed Program, which implements the equations in this application note, is available from Cypress for your use on personal computers.

Introduction (Continued)

Derivation of Applicable Equations

The charge, Q, stored on a capacitor, C, that is charged to a voltage, V, is given by the equation;

$$Q = CV. \qquad \textbf{EQ. 1}$$

Dividing both sides of equation 1 by the time required to charge and discharge the capacitor (one period or T) yields;

$$\frac{Q}{T} = \frac{CV}{T} \qquad \textbf{EQ. 2}$$

By definition, current (I) is the charge per unit time and

$$f = \frac{1}{T}.$$

Therefore,

$$I = CVf. \qquad \textbf{EQ. 3}$$

The power (P = VI) required to charge and discharge the capacitor is obtained by multiplying both sides of equation 3 by V.

$$P = VI = CV^2f \qquad \textbf{EQ. 4}$$

It is standard practice to make the assumption that the capacitor is charged to the supply voltage (V_{CC}) so that

$$P = V_{CC}I = C\,[V_{CC}]^2f \qquad \textbf{EQ. 5}$$

The total power consumption for a CMOS integrated circuit is dependent upon:

- the static (quiescent or DC) power consumption.
- the internal frequency of operation
- the internal equivalent (device) capacitance
- the number of inputs, their associated capacitance, and the frequency at which they are changing
- the number of outputs, their associated capacitance, and the frequency at which they are changing

In equation form:

$$P_D = [(C_{IN})(F_{IN}) + (C_{INT})(F_{INT}) + (C_{LOAD})(F_{LOAD})]\,[V_{CC}]^2 + I_{CC}\,(\text{quiescent})\,V_{CC}. \qquad \textbf{EQ. 6}$$

The first three terms are frequency dependent and the last is not. This equation can be used to describe the power dissipation of every IC in the system. The total system power dissipation is then the algebraic sum of the individual components.

The relative magnitudes of the various terms in the equation are device dependent. Note that equation 6 must be modified if all of the inputs, internal nodes or all of the outputs are not switching at the same frequency. In the general case, each of the terms is of the form C1 F1 + C2 F2 + C3 F3 + ... Cn Fn. In practical reality the terms are estimated using an equivalent capacitance and frequency.

Transient Power: Input Buffers and Internal

In the N-well CMOS inverter, the P-channel pullup transistor and the N-channel pulldown transistor (which are in series with each other between V_{CC} and V_{SS}) are never on at the same time. This means that there is no direct current path between V_{CC} and ground, so that the quiescent power is very nearly zero. In the real world, when the input signal makes the transition through the linear region (i.e., between logic levels) both the N-channel and the P-channel transistors are partially turned ON. This creates a low impedance path between V_{CC} and V_{SS}, whose resistance is the sum of the N-channel and P-channel resistances. These gates are used internally in Cypress products.

DC or Static Power

In addition to the conventional gates there are sense amplifiers, input buffers and output buffers, bias generators and reference generators that all dissipate power. The RAMs and FIFOs also have memory cells that dissipate standby power whether the IC is selected or not. The PROM and PAL® products have EPROM memory cells that do not dissipate as much standby power as a RAM cell.

Power Down Options

Many of the Cypress static RAMs have power down options that enable the user to reduce the power dissipation of these devices by approximately an order of magnitude when they are not accessed. The technique used is to disable or turn-off the input buffers and the sense amplifiers.

Worst Case Device Power Specifications

All Cypress products are specified with I_{CC} under worst, worst, worst case conditions. This means that the V_{CC} voltage is at its maximum (5.5V), the operating temperature is at its minimum, which is 0°C for commercial product and −55°C for military product and all inputs are at V_{IN} = 1.5V.

I_{CC} TEMPERATURE DEPENDENCE

For all Cypress products operating under all conditions, the I_{CC} current increases as the temperature decreases. The I_{CC} temperature coefficient is −0.12% per °C. To calculate the percentage change in I_{CC} from one temperature to another, this temperature coefficient is multiplied by the temperature difference.

If, for example, it is required to calculate the expected reduction in I_{CC} if either a commercial or a military grade Cypress IC is operated at room temperature (25°C), the calculations are:

For commercial products

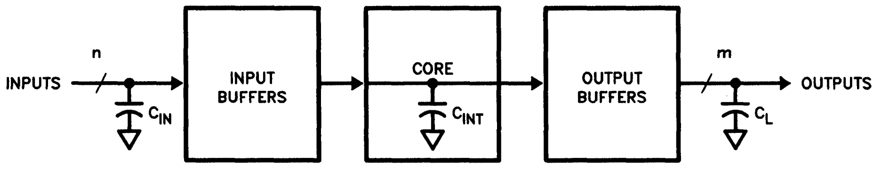
$[0 - 25] \times [-0.12\%] = 3\%$ less I_{CC} at room temperature than at 0°C.

For military products

$[-55 - (25)] \times [-0.12\%] = 9.6\%$ less I_{CC} at room temperature than at −55°C.

Procedure

The procedure will be to develop a general purpose power dissipation model that applies to all of the Cypress CMOS products and to then present tables so that users can estimate typical and worst case power dissipations for each product. The data will be presented in chart form as functions of product type and capacitance, that is: SRAM, PROM, PAL or Logic; including FIFOs.



0059-1

Figure 1. Power Dissipation Model

Power Dissipation Model

A general purpose power dissipation model for all Cypress integrated circuits is shown in Figure 1.

The procedure will be to isolate the four components of power dissipation described by equation 6 by controlling the inputs to the IC. The quiescent (I_{CC}) current is measured with the inputs to the IC at 0.4V or less. Under this condition the input buffers and output buffers (unloaded DC wise) draw only leakage currents. All other direct currents are due to the substrate bias generator, sense amplifiers, other internal voltage or current references and NMOS memory circuits.

At V_{IN} = 1.5V the input buffers draw maximum I_{CC} current. The total current is measured and the quiescent current subtracted to find the total input buffer I_{CC} current. The current per input buffer is then calculated by dividing the total input buffer current by the number of input buffers.

INPUT BUFFERS

Three different types of input buffers are used in Cypress products. For purposes of illustration they are referred to as types A, B and C. Table 1 lists the maximum ICCs.

Table 1. Types of Input Buffers

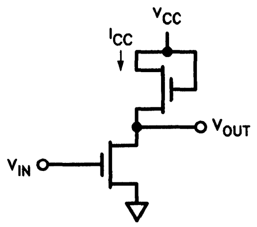
Buffer Type	I_{CC} (max. in mA)
A	1.3
B	0.8
C	0.6

The schematics and input characteristics for the three types of buffers are illustrated in Figure 2. A circle on the gate of a transistor means that it is a P-channel device.

As can be seen from the figure, the input buffers draw essentially zero I_{CC} current when V_{IN} is 0.4V or less or (except for type A) when V_{IN} is 4V or more. In other words, if the inputs are driven "rail to rail" the B and C input buffers will dissipate power only during the input signal transitions.

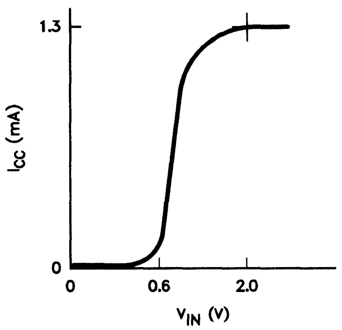
To reach these levels the input pins should be either driven by a CMOS driver or by a TTL driver whose output does not drive any other TTL inputs.

When the inputs are driven by the minimum TTL levels (V_{IH} = 2V, V_{IL} = 0.8V) each input buffer draws 20% more I_{CC} current than if it were driven rail to rail.



0059-3

Figure 2A



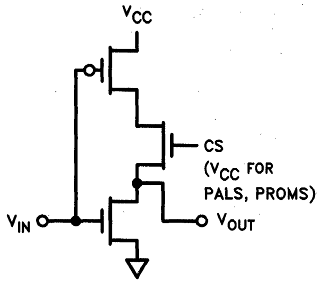
0059-4

Figure 2B
Type A

Power Dissipation Model (Continued)

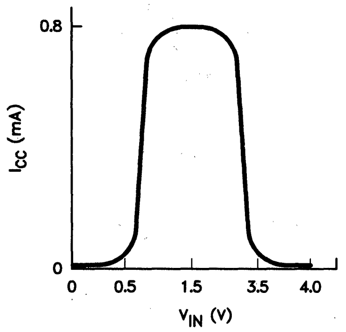
DUTY CYCLE CONSIDERATIONS

The input characteristics of the type B (Figure 2D) and the type C (Figure 2F) buffers may be approximated by triangles symmetric about the $V_{IN} = 1.5V$ points, whose amplitudes are 0.8 mA and 0.6 mA, respectively. Therefore, between the $V_{IN} = 0.5V$ and $V_{IN} = 3.5V$ points the average current is one-half the peak current, or 0.4 mA and 0.3 mA, respectively. In most systems the input signal slew rates are two volts per nanosecond or greater so the input transitions occur quickly. Under these conditions the duty cycle of the input buffers must be considered.



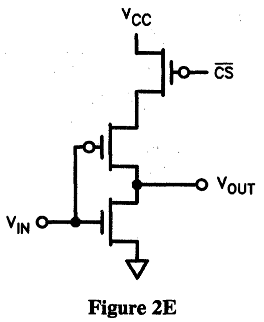
0059-5

Figure 2C



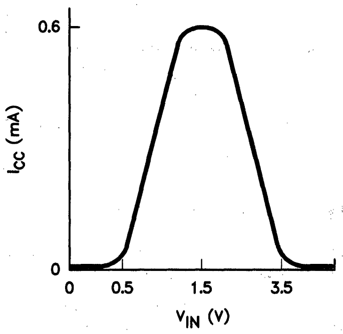
0059-6

Figure 2D
Type B



0059-7

Figure 2E



0059-8

Figure 2F
Type C

For example, if the CY7C167-35 RAM were used with input signals having a slew rate of two volts per nanosecond it would take

$$[3.5V - 0.5V] \times \frac{1}{2V/ns} = 1.5 \text{ ns}$$

for the input signals to go through the 3V transition. During the transition each input buffer would be drawing 0.3 mA of current from the I_{CC} supply. However, this time is only 1.5 ns/35 ns = 0.0429 or 4.29% of the access cycle. Therefore, the actual input buffer transient current is only 0.0429 × 0.3 mA = 0.01287 mA. It will be shown that this is insignificant in most power calculations.

INPUT BUFFER FREQUENCY DEPENDENT CURRENT

This is the current required to charge and discharge the capacitance associated with each input buffer. The capacitance is typically 5 pF and the voltage swing is typically 4V.

Using equation 3; $I = CVf$

$$I_{CC}(f) = 5 \times 10^{-12} \times 4 \times f.$$
$$I_{CC}(f) = 20 \times 10^{-12} f.$$

CORE AND OUTPUT BUFFERS

The memory array will have a standby power dissipation due to the substrate bias generator, reference generators, sense amplifiers, and polyload RAM cells or EPROM cells. This current is measured with $V_{IN} = 0V$, so that the input buffers draw no current. Under these conditions the output buffers will draw only leakage current and dissipate essentially no power.

The output buffers have N-channel pullup devices that cause the output voltage level to reach $V_{OH} = V_{CC} - 1V$.

The capacitance of the output buffers, including stray capacitance, is typically 10 pF.

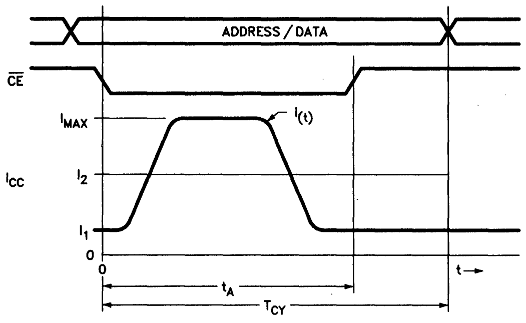
$$\text{If } C_L = 10 \text{ pF}, V_{OH} \cong 4V.$$

Again, using equation 3, $I_{CC}(f) = 40 \times 10^{-12} f$ for the output buffers.

Current Measurement

INSTANTANEOUS CURRENT

Figure 3 illustrates the instantaneous current drawn by a Cypress RAM. The instantaneous power is calculated by multiplying this current times the constant supply voltage, V_{CC}. Most of the power is dissipated in the time corresponding to the access time. This is also true for PROMs and PALs.



I_1 = Quiescent I_{CC}
I_2 = Average I_{CC}
$i(t)$ = Instantaneous I_{CC}

Figure 3. RAM I_{CC}

AVERAGE CURRENT

The current measurement unit in an automatic tester integrates the instantaneous current over the measurement cycle and arrives at an equivalent average current. In other words, the average current, I_2, during time TCY is equal to the area between the instantaneous current, i (t), and the X axis during TCY. Therefore, when the frequency is decreased, the "current pulse" is (figuratively) spread over a longer time, so the average current is proportionately less.

DC Load Current

Note that the preceding calculations have not accounted for any DC loads. The user must calculate these separately.

Product Characteristic Tables

The following tables are listed to enable the user to calculate the current requirements for Cypress products. C_{INT} is the equivalent device internal capacitance, I_{CC} (Q) is the quiescent or DC current and $I_{CC(MAX)}$ is the maximum I_{CC} current (as specified on the data sheet) for the commercial operating temperature range. Conditions are V_{CC} = 5V and T_A = 25°C.

STATIC RAMs

Table 2

Part No.	Buffer Type	No. Inputs	No. Outputs	C_{INT} (pF)	I_{CC} (Q) (mA)	I_{CC} (Max.) (mA)
CY7C122/123	A	16	4	24	50	90
CY7C128	B	14	8	27	59	120
CY7C147	B	15	1	34	28	90
CY7C148/149	B	12	1	32	45	90
CY7C150	B	18	4	20	44	90
CY7C161/162	B	22	4	300	13	70
CY7C164	B	20	4	300	13	70
CY7C166	B	21	4	300	13	70
CY7C167	C	17	1	75	25	70
CY7C168/169	C	18	4	75	50	70
CY7C170	B	18	4	50	33	90
CY7C171/172	B	18	4	100	27	70
CY7C185/186	B	25	8	330	13	100
CY7C187	B	19	1	150	7	100
CY7C189/190	B	10	4	21	32	90

PROMs

Table 3

Part No.	Buffer Type	No. Inputs	No.* Outputs	C_{INT} (pF)	I_{CC} (Q) (mA)	I_{CC}(Max.) (mA)
CY7C225	B	12	8	32	35	90
CY7C235	B	13	8	35	35	90
CY7C245	B	13	8	35	50	90
CY7C251	C	18	8	43	9.5	100
CY7C254	C	18	8	43	35	100
CY7C261/3/4	C	14	8	60	45	100
CY7C268	C	19	1/8	60	60	100
CY7C269	C	17	1/8	60	60	100
CY7C281/282	B	14	8	35	35	100
CY7C291/292	B	14	8	35	50	100

*/Bidirectional pins

PALs

For the 16L8, 16R8, 16R6 and 16R4 the number of inputs and outputs is, within limits, user configurable. All use type B buffers.

Table 4

Part No.	C_{INT} (pF)	I_{CC} (Q) (mA)	I_{CC}(Max.) (mA)
PALC16L8/R8/R6/R4	40	25	45
PLDC20G10	50	30	55
PALC22V10	50	40	80
PLDCY7C330	300	42	120

LOGIC PRODUCTS

Table 5

Part No.	Buffer Type	No. Inputs	No.* Outputs	C_{INT} (pF)	I_{CC} (Q) (mA)	I_{CC}(Max.) (mA)
CY7C401	B	6	6	53	30	75
CY7C402	B	7	7	53	30	75
CY7C403	B	7	6	53	30	75
CY7C404	B	8	7	53	30	75
CY7C408	B	11	12	100	42	135
CY7C409	B	11	13	100	42	135
CY7C428/9	C	14	12	190	18	80
CY7C510	C	24	19/16	60	30	100
CY7C516	C	28	16/16	60	30	100
CY7C517	C	28	16/16	60	30	100
CY3341	B	6	6	53	30	45
CY7C601	C	25	19/64	950	89	600

Product Characteristic Tables (Continued)

Table 5 (Continued)

Part No.	Buffer Type	No. Inputs	No.* Outputs	C_{INT} (pF)	I_{CC} (Q) (mA)	$I_{CC(Max.)}$ (mA)
CY7C901	C	24	10/4	160	25	80
CY7C909	C	21	5	80	25	55
CY7C910	C	22	16	150	2.6	70
CY7C911	C	13	5	80	25	55
CY7C9101	C	36	22/4	70	30	60
CY7C9116	C	22	1/20	1000	35	150
CY7C9117	C	38	1/4	1000	35	150

*/Bidirectional pins

Static RAM Example

To illustrate how to use the preceding tables and perform the required calculations the following example is provided.

Estimate the typical I_{CC} current for the CY7C169-35 RAM at room temperature (T_A = 25°C) and V_{CC} = 5V. Assume the duty cycle is 100% at the specified access time. Calculate typical and worst case I_{CC} (all inputs and outputs changing) with output loading of 10 pF.

From the RAM product characteristic table;

# inputs = 18

# outputs = 4

C_{INT} = 75 pF

I_{CC} (Q) = 50 mA

TRANSIENT INPUT BUFFER CURRENT

The input buffers on the CY7C169 are type C, so the average current is 0.3 mA. If the input signal level transitions are 4V and the transition times are 2 V/ns, the transition time is:

$$Tt = \frac{4V}{2\ V/ns} = 2\ ns.$$

The duty cycle is then;

$$2\ ns/35\ ns = 0.057.$$

Therefore, each input buffer draws

$$0.3\ mA \times 0.057 = 0.0171\ mA.$$

If all inputs change, the total transient input buffer current is

$$18 \times 0.0171 = 0.31\ mA.$$

CVf Input Buffer Current

I = CVf C_{IN} = 5 pF

I = 0.57 mA V = 4V

f = 1/35 ns

Total = 18 × 0.57 = 10.28 mA

Internal CVf Current

I = CVf C_{INT} = 75 pF

I = 10.71 mA V = 5V

f = 1/35 ns

Output CVf Current

I = CVf C_{OUT} = 10 pF

I = 1.15 mA V = 4V

f = 1/35 ns

Total = 4 × 1.15 = 4.6 mA

The Quiescent Current is 50 mA

The Total Current At TCY = 35 ns is;

Input Transient	0.31 mA
Input CVf	10.28 mA
Internal CVf	10.71 mA
Output CVf	4.6 mA
Quiescent	50 mA
Total I_{CC}	75.9 mA (all inputs/outputs changing)

Note that the worst case transient current is 25.9 mA.

If one-half of the inputs and outputs change this is reduced to 12.95 mA, which gives a total current of 63 mA (typical I_{CC}).

If the duty cycle is 10% the transient current is reduced to 1.3 mA, which results in a total current of 51.3 mA.

Note also that the Input CVf current and the output CVf current would have the same values for a bipolar device.

WORST, WORST, WORST CASE I_{CC}

Next, let's estimate the I_{CC} for worst case V_{CC} and low temperature, in addition to all inputs and outputs changing and compare it with the I_{CC} specified on the data sheet.

The I_{CC} current will be greater at high V_{CC}, which is 5.5V or 1.1 × the nominal 5V V_{CC}. The increase in I_{CC} due to the lower temperature is 3%, so the total increase is 13%. These factors apply to the internal CVf current (10.71 mA), the output CVf current (4.6 mA), and the quiescent current (50 mA), (total 65.31 mA).

Total I_{CC} = Input Transient I_{CC} + Input CVf I_{CC} + [Internal CVf + Output CVf + I_{CC}(Q)] × 1.13

I_{CC} = 0.31 + 10.28 + [65.31] × 1.13 = 84.4 mA.

This is approximately 94% of the 90 mA specified on the data sheet.

Note, however, that the data sheet I_{CC} maximum does NOT include the output CVf current.

Typical I_{CC} Versus Frequency Characteristic

The I_{CC} versus frequency curves for all Cypress products have the same basic shape, which is illustrated by the PAL 16R8 curve of Figure 4. The current remains essentially constant at the quiescent I_{CC} value until the frequency increases to the point where the capacitances begin to cause appreciable currents. This point depends upon the capacitances (input, internal, and output), the number of inputs and outputs, the rate at which they change, and the voltage levels that they are switched between. For Cypress products this point is in the 1–10 MHz range.

Typical I_{CC} Versus Frequency Characteristic (Continued)

The PAL 16R8 devices that were tested to obtain the data for the curve were exercised such that all inputs and all outputs changed every cycle. Curve A shows the total I_{CC} current for a 50 pF load on each of the eight outputs. Curve B shows the total I_{CC} current when the outputs are disabled. The B curve results from the input and the internal capacitances. In most applications the actual operation of the device will be somewhere between the A and B curves.

The A and B curves may be extrapolated backwards until they intersect the quiescent current (point C in Figure 4). Point C is approximately 5.6 MHz. This gives the user an easy to use approximate formula to calculate the I_{CC} current.

For frequencies less than 5.6 MHz

$$I_{CC} = I_{CC}(Q) = 25\text{ mA}$$

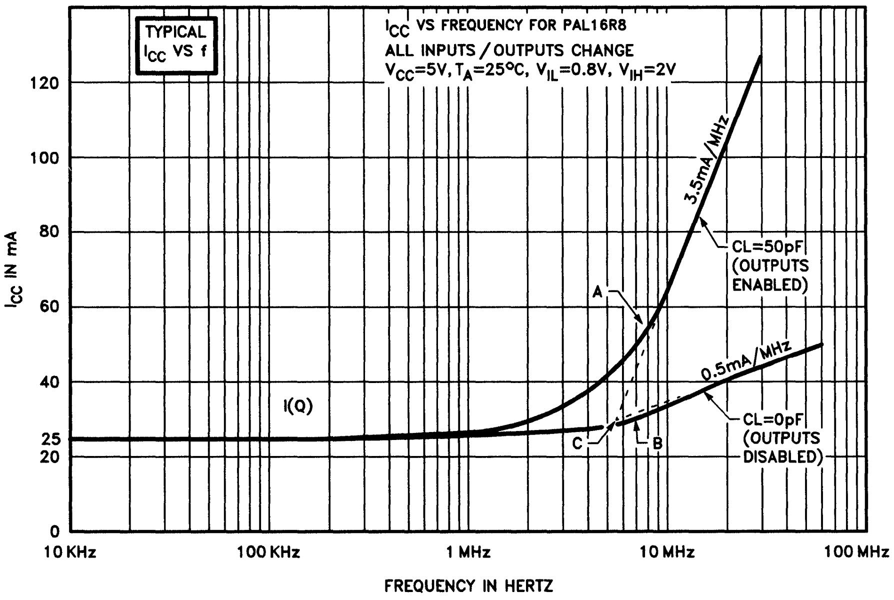
For frequencies greater than 5.6 MHz

$I_{CC} = I_{CC}(Q) + 3.5$ mA per MHz (all outputs changing)

or,

$I_{CC} = I_{CC}(Q) + 0.5$ mA per MHz (no outputs changing)

Frequency in Hertz



0059-9

Figure 4. Typical I_{CC} vs f

NOTES:

Section Contents

CYPRESS SEMICONDUCTOR

Tips for High-Speed Logic Design

Introduction

As electronic system clock rates reach ever higher, logic designers who were engineering 10 MHz, 100 nsec cycle time systems are recently finding themselves working with 20 MHz, 50 nsec cycle time (and faster) systems. These same designers are discovering that the techniques that worked fine at 10 MHz are no longer appropriate at 20 MHz and beyond. At 10 MHz, one can utilize sluggish and relatively well-behaved LS TTL logic with its leisurely setup and hold parameters, long propagation delays, forgiving output enable and disable times, and high-output current drive capacity. As clock rates cranked up, designers turned to faster bipolar logic families, but found that power dissipation rose proportionally. To save power and enhance reliability, modern electronic engineers are switching to CMOS components, and have been happy to find that CMOS can deliver the speed they require at the low power levels they desire. In the quiescent state, CMOS logic (AC/ACT/FCT) draws three to five orders of magnitude less power than bipolar logic (LS/ALS/AS). At 1 MHz, CMOS logic dissipates about 0.1 mW per gate, while LSTTL logic dissipates about 2.0 mW per gate. CMOS technology has truly rewritten the speed/power rules set forth in the bipolar era.

However, there are still plenty of challenges that face the high-speed logic designer. High-performance logic families are sensitive to system noise and are also noise generators themselves. As a result of the effort to make these devices as speedy as possible, they often have anemic output drive capacity. Clock distribution becomes much more of an issue at high frequencies because skew and slow rise times degrade operating margins. As bus cycles tighten up, it becomes more and more difficult to avoid bus clashes (multiple devices driving a bus). Very fast SRAMs and FIFOs require read and write pulse widths that are very difficult to synthesize using synchronous logic (hence the appearance of self-timed memory devices). PLDs have become ubiquitous in modern board-level designs, but their relatively long propagation delays and slow switching speeds need to be carefully considered by high-speed designers. Printed circuit boards can no longer be thought of as an ideal electrical interconnect. In the high-speed realm, the effects of distributed capacitance, inductance, and propagation delay on the PCB must be taken into account. The resistive termination of critical signals to mitigate the effects of ringing becomes a practical necessity above 20 MHz. In the days of old, it wasn't appropriate to factor loading into propagation delays. Today, the conservative designer accounts for loading when calculating worst-case prop delays and worst case signal skew. Heavy capacitive bypassing and low inductance decoupling is essential to minimize switching noise above 20 MHz. Metastability, a phenomenon not widely appreciated until recently, is a critical issue in high-frequency systems. It is essential to be able to resolve asynchronous events quickly and reliably in high-performance designs. Finally, crosstalk is a substantial concern with high slew rate and noise sensitive CMOS logic.

This application note provides tips and makes substantive suggestions for designing high speed logic circuits that operate reliably. The tips and suggestions are loosely organized under the following headings:

Noise Considerations
Clock Distribution
Buses and Memories
Care and Feeding of PLDs
PCB Effects
Metastability and Crosstalk

Noise Considerations

High-speed CMOS logic tends to be noisier than LSTTL because CMOS voltage swings are rail-to-rail and because of the faster edge rates (2 volts per nsec and faster) made possible by small geometry, dual-layer metal CMOS technology. The classic ground bounce noise situation arises when several outputs of a CMOS logic device are switching from the high state to the low state. The simultaneous switching causes a relatively large sink current from the load capacitance to flow to ground through the device package inductance. A potential is momentarily developed across this inductance that is equal in magnitude to the product of the package inductance and the time rate of change of the sink current. This ground bounce voltage spikes the low voltage state held on the quiescent outputs, and this spike can often exceed the input low-level maximum voltage (0.8 V), causing the downstream logic device to switch erroneously. It turns out that both the chip ground reference and the chip Vcc reference are spiked, but because more energy is switched through the ground lead inductance, it is much more common to see a problem in a quiescent low-state output. What can be done to minimize ground and Vcc bounce noise?

1. Any steps which will reduce the parasitic inductance between the package and ground and Vcc should be pursued. This includes using a PCB with ground and Vcc planes or at the very least power distribution elements, avoiding the use of sockets, and using low inductance decoupling and bypass capacitors. On critical parts, use a standard ceramic decoupling cap (0.01 to 0.1 uF) along with a high-frequency decoupling cap (approximately 470 pF). The Rogers Corp. Micro/Q 1000 Series High Frequency low inductance caps are optimal for this purpose. Surface mount packages have lower package inductance than DIP packages. So called "rotated die" devices with center Vcc and ground pins also have lower inductance.

2. Whenever possible, design synchronous circuits. The ground bounce produced by a octal register, for instance, is triggered by the clock. If the register is feeding another registered device, then the noisey output have until a setup time before the next clock to settle. When compelled to drive an asynchronous signal with an octal driver, use an output pin close to the package ground pin. The output pin next to the Vcc pin can have as much as 50% more ground bounce noise than the output pin next to the ground pin.

3. Various techniques can be used to slow the switching or transition edge rates and, therefore, the time rate of change of the sink and source currents. It can be accomplished with series damping resistors, or by increasing the inductance or capacitance between the output pin of the driving device and the input pin of the receiving device. Printed circuit board traces possess parasitic ground path capacitance and inductance which are trace length and trace topology dependent and thus difficult to predict. The most common technique is the use of series damping resistors, in the 25 to 35 ohm range (33 ohms is a standard value). Series resistors also limit signal overshoot and undershoot.

4. Try to avoid running control signals through a device that drives data and address lines. When using a 10 output PLD (such as a 22V10) in an 8-bit bus oriented application, it is tempting to use the extra two outputs for control signals. It is very likely that these control lines will be disturbed if the other eight lines are simultaneously switched. Using devices that feature input hysterisis will add to the noise margin. Input hysterisis can typically provide 200 mV of additional noise immunity.

Mixing logic families can compromise noise immunity margins. For comparison purposes, the margin for a particular logic family is the magnitude difference between guaranteed input threshold of the family and the guaranteed output voltage for the high and low states, i.e., |Vil - Vol| / |Vih - Voh|.

When possible, use a logic family that can drive 50 ohm (commercial) transmission lines directly. This specification is characteristic of devices that can switch sufficient current to guarantee so called "incident wave" switching. Switching that occurs on the incident wave is obviously faster than having to wait for the reflected wave.

In addition to causing false triggering of downstream sequential logic and glitches in downstream combinatorial logic, ground bounce noise can also cause registers in the bounced device to "forget" their stored state. This is due to the momentary disturbance in the chip's ground and Vcc reference. The switching of multiple outputs also has the effect of skewing the device's propagation delay, approximately 200 psec per switched output. With an octal or ten bit device, this 1 to 2 nsec additional delay should be included in worst case timing analyses.

Clock Distribution

Adequate clock distribution is essential when designing 20 MHz and faster systems because skew can eat up precious nanoseconds and because high-speed logic devices are very sensitive to clock waveform distortion and slow rise times. All physical devices exhibit an edge-dependent propagation delay asymmetry, i.e., a low-to-high going edge will propagate more quickly than a high-to-low going edge, or vice versa. For example, the clock to Q prop delay for a 74F74 from Signetics ranges from 3.8 nsec to 6.8 nsec low-to-high, and 4.4 nsec to 8.0 nsec high-to-low. The 74AS1000 NAND driver from Texas Instruments specs a 1 to 4 nsec range for both low-to-high and high-to-low edges, but any particular physical device will show some asymmetry. It is possible to maintain duty cycle symmetry in a buffered clock distribution network by cascading two inverting drivers. The two drivers must both be in the same package, as shown in *Figure* 1.

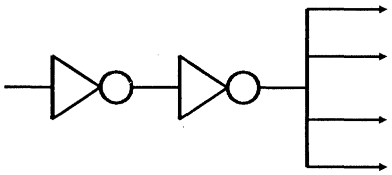


Figure 1. Maintaining Duty Cycle Symmetry

Because the two drivers are in the same package, their prop delay characteristics will track, and the high-to-low and low-to-high differential delays will tend to cancel.

The fanout from a clock buffer should be limited to 8 to 15 devices. Fanout calculations must account for both AC and DC loading. The AC characteristics for logic components are specified at 50 pF of load capacitance, and occasionally at 300 pF of load capacitance. Propagation delays and output enable times increase by approximately 1 nsec per each 50 pF of additional load capacitance. The input capacitance of bipolar logic families is higher (approximately 10 pF) than that of CMOS (approximately 5 pF). If the sum of the capacitance being driven exceeds 50 pF, the AC characteristics of the driver should be derated appropriately.

The important DC electrical characteristic for the purposes of loading is input current. The driving device must be able to sink the sum of the low-level input currents to which it is connected (Iol at Vol). The driving device must also be able to source the sum of the high-level input currents to which it is connected (Ioh at Voh). The low-level input current for bipolar logic families ranges from -400 uA to -100 uA, while the low-level input current for modern CMOS logic families ranges from -5 uA to -1 uA. The high-level input current for bipolar logic families ranges from 50 uA to 20 uA, while the high-level input current for modern CMOS logic families ranges from 5 uA to 1 uA. Since the Iol at Vol for bus drivers is often as high as 48 mA, and the Ioh at Voh is often as high as -24 mA, input current loading is seldom an issue except when driving a parallel (resistor) terminated load. For example, a 220 ohm pullup requires about 22 mA worst case (Vol = 0V, Vcc = 5V), and a 330 ohm pulldown requires about 15 mA worst case (Voh = 5V, Gnd = 0V). Consider using an AC termination scheme if this additional current cannot be tolerated.

When a clock fanout greater than that which can be safely supplied by a single buffer is required, parallel drivers should be used *Figure* 2.

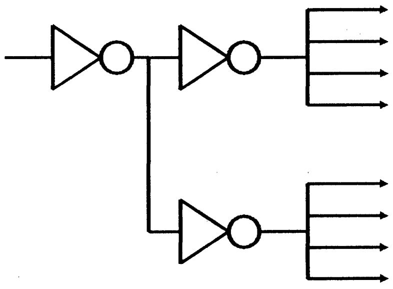


Figure 2. Parallel Clock Drivers

When distributing a clock signal, attempt to load each of the parallel lines equally. Unequal loading will increase the skew between lines.

Buses and Memories

When designing buses in high-performance systems, it is important to consider the effects of AC and DC loading as discussed above. The input and output capacitance of CMOS SRAMs, PROMs, and DRAMs ranges between 5 and 7 pF. This can become a concern with large memory arrays. Be especially careful when using SRAM modules, which can have high input and output capacitances due to the multiple devices connected to each signal line. Because the signals that drive large memory arrays (such as the address, RAS, CAS and data lines) tend to have long PCB traces, it is common practice to series terminate these lines to minimize ringing, undershoot, and overshoot. The input load or leakage currents for CMOS SRAMs, PROMs, and DRAMs is approximately 10 uA, sink and source. When high-output-current bus drivers are used (24 mA Iol or greater), DC loading is rarely an issue.

As system cycle times shorten, it becomes more difficult to avoid bus clash situations. Bus clash or bus contention occurs when, on a shared bus, one tri-state device finishes its output enable time before a second device finishes its output disable time. For a short period of time both devices are driving the bus. Because the output stages of memories and logic components can typically withstand at least 20 mA of current, the excess current won't damage the useful life of the device. The problem with bus clash is that it causes large positive and negative current changes in the device Vcc and ground paths. This demand for current induces Vcc and ground bounce noise just like the simultaneous switching situation previously discussed. An overlap in the worst case output enable and output disable times of greater than 5 nsec should be avoided.

The fact that CMOS components draw very little input current can be used to advantage on busses when hold time is deficient. For example, consider the situation when a CMOS memory is connected to a CMOS octal register. The memory is read, the /OE (or the /CE) is deasserted, and the data is clocked into the register. Ordinarily, the data should be clocked into the register before /OE is deasserted since the output disable time for the memory could be very short (worst case). However, when the memory was read, the distributed capacitance presented by the register inputs, the PCB trace, and its own outputs was charged. Because the output leakage current of the memory and the input current of the register are very low (5 to 10 uA), this distributed capacitance remains charged for some time, and the data is in effect held long enough to make up for the deficient timing.

High-speed SRAMs and FIFOs have timing requirements that are often difficult to meet using synchronous circuits. In such situations, there are asynchronous alternatives to consider. Various manufacturers supply delay lines, the output taps of which can be combinatorially gated to synthesize the required signal. Delay lines are typically calibrated by comparing the rising edge of the input to the rising edges of the various delayed outputs; the delay times for the falling edges are less accurate. If a decoded signal uses falling edges, make sure that the design can tolerate a few nanoseconds of slop. The Engineered Components Company makes a family of pulse generator modules (PGMs), which issue a precise pulse when presented with a positive going edge. They offer standard PGMs, fast-recovery PGMs that have a higher maximum repetition rate, and delayed PGMs which wait a specified period before issuing the pulse. Both delay lines and PGMs have propagation delays that range from 5 to 10 nsec.

Care and Feeding of PLDs

Programmable Logic Devices (PLDs) are exceedingly useful for designing high-performance systems, but their characteristics and shortcomings must be well understood. The set-up time for most registered PLDs is usually just less than the propagation delay. This is because the signal to be latched must propagate through the AND array as well as the OR/XOR gate before reaching the flip-flop, while the clock is connected directly from the pin to the flip-flop. Accordingly, the hold time for this type of PLD is 0 nsec minimum worst case and several nanoseconds negative typically. This "negative hold time" implies that the PLD samples the state of the inputs as they existed several nanoseconds before the rising edge of the clock. This phenomenon can be used to advantage when the device feeding the PLD is hold-time deficient with respect to the PLD clock.

Beware of slow rise and fall times on signals generated by PLDs. PLD outputs aren't as quick and don't have the drive capacity of standard logic. When generating a critical signal, such as a FIFO read or shift out pulse in a PLD, buffer it with a fast, hard-driving gate. Identical equations in the same PLD can exhibit different propagation delays due to nonidentical on-chip path lengths. PLD propagation delays are especially dependent on capacitive loading.

PCB Effects

The most conservative approach to handling the signal distortion effects of Printed Circuit Boards (PCBs) is to consider every substrate interconnect as a transmission line. In practice, this conservative approach only works when the unloaded signal transition time approaches the round-trip substrate propagation delay. For ordinary PCB materials (G-10 fiberglass epoxy), the round trip propagation delay is approximately 0.3 nsec per inch. Therefore, for 3 nsec transition times, any PCB trace longer than 10 inches should be considered a transmission line. A transmission line presents a characteristic impedance and possesses distributed inductance and capacitance. Ringing on a transmission line is minimized when the output impedance of the driving device is closely matched to the characteristic impedance of the line. The theoretical unloaded characteristic impedance of a 10 mil wide, 1 oz. copper line (1.5 mils thick) over a ground plane separated by a dielectric of G-10 fiberglass epoxy 62.5 mils thick is approximately 130 ohms (microstrip model). In reality, PCB trace characteristic impedances can range from 50 to 200 ohms. Capacitive loading reduces the characteristic impedance, increases the delay, and slows the rise time on a transmission line.

The conventional method for reducing reflections on transmission lines is with some form of termination, the most common being the so-called Thevenin type consisting of a pullup resistor to Vcc and a pulldown resistor to ground. The goal is to match the Thevenin equivalent of the two resistors to the characteristic impedance of the trace. Common values are 220 ohms pullup and 330 pulldown, which yields a Thevenin equivalent of 132 ohms, and 330 ohms pullup and 470 ohms pulldown, which yields a Thevenin equivalent of 194 ohms. Both of these termination pairs pull the line to logic high (approximately 3V) when the driver is disabled. The termination resistors should be placed as close as possible to the receiver. Keep in mind that many CMOS logic components have input and output clamp diodes to help damp overshoot and undershoot.

Metastability and Crosstalk

The output of a latch or flip-flop can go into an undefined or metastable state (neither logic high or logic low) when the setup time or hold time for the device is violated. The metastable condition typically occurs when an asynchronous signal is being synchronized. It occurs in all process technologies and is impossible to completely eliminate. The important parameters for the board designer to have are the Mean Time Between Failures (MTBF) at maximum operating frequency, and the average or typical time it takes the device to resolve from a metastable state to a stable state (resolution or settling time, Tsw). These parameters and/or the equations for deriving them should be available from the particular device's manufacturer. Metastability performance is proportional to the Vih to Vil slew time of a technology. High-speed CMOS registers such as those found in the PLDs made by Cypress have very fast slew times and typical settling times that range from 182 psec to 592 psec depending on the device type.

The double latching of asynchronous inputs is recommended to dramatically increase the MTBF of a system and reduce the probability of a metastable event causing system malfunctions. When determining the length of time to delay before clocking the second register, multiply the published typical settling time by two or three to create an extra margin of protection.

Crosstalk is the undesirable coupling of a transition on an active line (talker) on an inactive line (listener). The crosstalk amplitude is proportional to the talker edge rates, the physical proximity between signal lines, and the distance over which the two lines are parallel or adjacent. There are two important physical causes of crosstalk: mutual impedance and velocity differences. Mutual impedance is due to the mutual inductance and mutual capacitance between adjacent signal lines, and is basically a transformer-like effect. Velocity differences arise when a signal propagates along a conductor that is in contact with two materials of differing dielectric constants, such as fiberglass epoxy and air in PCBs. The wave propagating at the copper to fiberglass epoxy interface travels slower than the wave propagating at the copper to air interface. A pulse is developed that is equal in duration to twice the difference in arrival times of the two waves, and hence the magnitude of the disturbance increases when the length of the parallel or adjacent traces increases.

The two types of crosstalk are forward and reverse. Forward crosstalk occurs when the talker driver and the listener driver are at the same end of the signal line. Reverse crosstalk occurs when the talker driver and the listener receiver are at the same end of the signal line. Forward crosstalk is the result of both velocity differences and mutual impedance phenomena, while reverse crosstalk is the result of the mutual impedance phenomenon exclusively. Due to the fast edge rates of CMOS logic, crosstalk is a legitimate concern. The following steps can be taken to reduce forward and reverse crosstalk:

1. Maximize the distance between traces and minimize the length that traces are parallel or adjacent. When possible, the signals on adjacent PCB layers should be perpendicular. Use the power and ground layers as shields between the signal layers. On two-layer PCBs run ground lines between adjacent, parallel signal lines.

2. Make adequate provision when using flat ribbon cable to have every other conductor a ground line. Protect critical signals such as clock lines with a dedicated ground strip on PCBs or with a ground twisted pair on backplanes.

3. Thevenin termination of a line to its characteristic impedance will reduce the crosstalk amplitude by 50%.

Conclusion

This applications note has attempted to provide the designer of high-speed digital systems with tips and advice to avoid some of the pitfalls that can arise. It is hoped that the reader will be in a better position to design and debug high-speed systems armed with the information provided in this note.

CYPRESS SEMICONDUCTOR

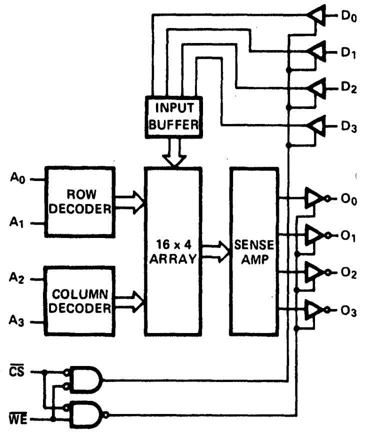
Application Briefs RAM Input Output Characteristics

Introduction to Cypress RAMs

Cypress Semiconductor Corporation uses a speed optimized CMOS technology to manufacture high speed static RAMs which meet and exceed the performance of competitive bipolar devices while consuming significantly less power and providing superior reliability characteristics. While providing identical functionality, these devices exhibit slightly differing input and output characteristics which provide the designer opportunities to improve overall system performance. The balance of this application note describes the devices, their functionality and specifically their I/O characteristics.

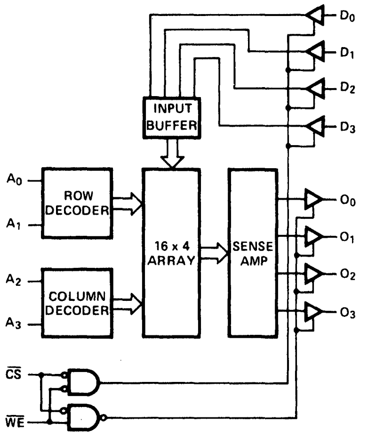
PRODUCT DESCRIPTION

The five parts in *Figure 1* constitute three basic devices of 64, 1024 and 4096 bits respectively. The 7C189 and 7C190 feature inverting and non-inverting outputs respectively in a 16 x 4 bit organization. Four address lines address the 16 words, which are written to and read from over separate input and output lines. Both of these 64 bit devices have separate active LOW select and write enable signals. The 256 x 4 7C122 is packaged in a 22 pin DIP, and features separate input and output lines, both active LOW and active HIGH select lines, eight address lines, an active LOW output enable, and an active LOW write enable. Both the



0027-1

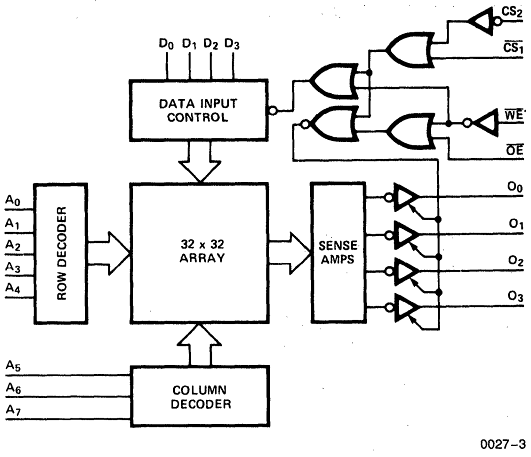
7C189



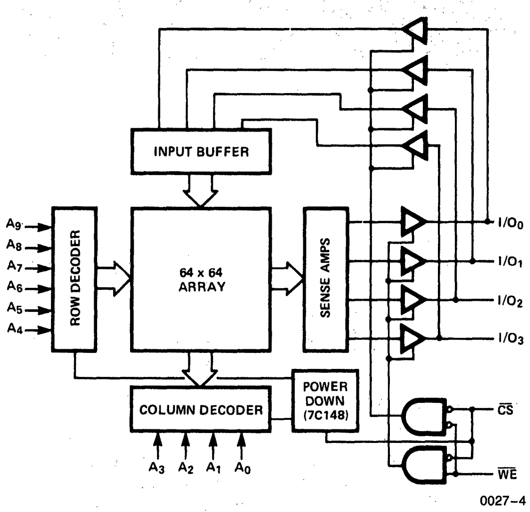
0027-2

7C190

Figure 1. RAM Block Diagrams



7C122



7C148/9

Figure 1. RAM Block Diagrams (Continued)

7C148 and 7C149 are organized 1024 x 4 bits and feature common pins for the input and output of data. Both parts have 10 address lines, a single active LOW chip select and an active LOW write enable. The 7C148 features automatic power down whenever the device is not selected, while the 7C149 has a high speed, 15 ns, chip select for applications which do not require power control. This family of high speed static RAMs is available with access times of 15 to 45 ns with power in the 300 to 500 mW range. They are designed from a common core approach, and share the same memory cell, input structures and many other characteristics. The outputs are similar, with the exception of output drive, and the common I/O optimization for the 7C148 and 7C149. For more detailed information on these products, refer to the available data sheets.

GENERIC I/O CHARACTERISTICS

Input and output characteristics fall generally into two categories, when the area of operation falls within the normal limits of V_{CC} and V_{SS} plus or minus approximately 600 mV, and abnormal circumstances, when these limits are exceeded. Inputs under normal operating conditions are voltages that switch between logic "0" and logic "1". We will consider operation in a positive true environment and therefore a logic "1" is more positive than a logic "0". The I/O characteristics of the devices we are concerned with are what is considered to be TTL compatible. Therefore a logic "1" is 2.0V, while a logic "0" is 0.8V. The input of a device must be driven greater than 2.0V, not to exceed V_{CC} + 0.6V to be considered a logic "1" and, to less than 0.8V, but not less than V_{SS} − 0.6V, to be considered a logic "0".

Output characteristics represent a signal that will drive the input of the next device in the system. Since the levels we are dealing with are TTL, we may assume that the V_{IL} and V_{IH} values of 0.8 and 2.0V referenced above are valid. In consideration of noise margin however, driving the input of the next stage to the required V_{IL} or V_{IH} is not sufficient. Noise margins of 200 to 400 mV are considered more than adequate, and therefore the V_{OH} we deal with is 2.4V while the V_{OL} is 0.4V, providing a noise margin of 400 mV. Since the driven node consists of both a resistive and a capacitive component, output characteristics are specified such that the output driver is capable of sinking I_{OL} at the specified V_{OL}, and capable of sourcing I_{OH} at V_{OH}. Since the values of I_{OL} and I_{OH} differ depending on the device, these values are shown in Table 1. Outputs have one other characteristic that we need to be concerned with, Output Short Circuit Current or I_{OS}. This is the maximum current that the output will source when driving a logic "1" into V_{SS}. We need to be concerned for two reasons. First, the output should be capable of supplying this current for some reasonable period of time without damage, and second, this is the current that charges the capacitive load when switching the output from a "0" to a "1" and will control the output rise time.

Since memories such as these are often tied together, we are also concerned about the output characteristics of the devices when they are deselected. All of the devices in this family feature three state outputs such that in addition to their active conditions when selected, when deselected, the outputs are in a high impedance condition which does not source or sink any current. In this condition, as long as the input is driven in its normal operating mode, it appears as an open, with less than 10 μA of leakage. Thus to any other device driving this node, it is non-existent.

TECHNOLOGY DEPENDENCIES AND BENEFITS

Some of the products in this application note were originally produced in a BIPOLAR technology, some have since been re-engineered in NMOS technology and Cypress has now produced them in a speed optimized CMOS technology. There are both technology dependencies and benefits relative to the design of input and output structures that are associated with each technology. The designer who uses these products should be knowledgeable of these characteristics and how they can benefit or impede a design effort. One of the most obvious is that both NMOS and CMOS device inputs are high impedance, with less than 10 μA of input leakage. Bipolar devices, however, require that the driver of an input sink current when driving to V_{IL}, but appear as high impedance at V_{IH} levels. This is due to the fact that the input of a bipolar device is the emitter of a bipolar NPN type device with its base biased positive. The bias is what establishes the point at which the input changes from requiring current to be sourced to high impedance and is 1.5V. This switching level is the reason that AC measurements are done at the 1.5V level. Although NMOS and CMOS device inputs do not change from low to high impedance, great care is taken to balance their switching threshold at 1.5V. To a system designer this allows fanout to consider only capacitive loading with MOS devices while bipolar has both a capacitive and DC component. The other input characteristic which differs from bipolar to MOS is the clamp diode structure. This structure exists in both MOS and bipolar, however in MOS that uses BIAS GENERATOR techniques, all high speed MOS devices, the diode does not become forward biased until the input goes more negative than the substrate bias generator plus one diode drop. Since the bias generator is usually about -3V this has the effect of removing the clamping effect.

I/O Parameters

CMOS/NMOS/BIPOLAR INPUT CHARACTERISTICS

Although NMOS, CMOS and BIPOLAR technologies differ widely, the I/O characteristics tend to fall into two areas. The traditional characteristics are the TTL derivatives that have been covered above, and are documented in Table 1. With the exception of the differences in input impedance between MOS and BIPOLAR devices all three technologies are used to produce TTL compatible products. The second camp is the true CMOS interface where signals swing from V_{SS} to V_{CC}. These interface specifications define a "1" as greater than $V_{CC} - 1.5$V and a "0" as less than $V_{SS} + 1.5$V. In addition, loads are primarily capacitive. Only devices produced in a CMOS technology are capable of behaving in this manner. CMOS devices can, however, handle both TTL and CMOS inputs. Devices such as the ones described in this application note have input characteristics depicted in *Figure 2.*

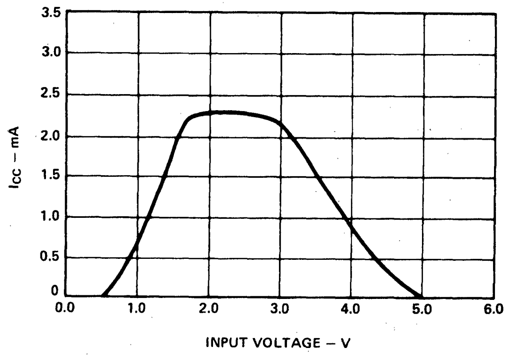


Figure 2. Input Voltage vs. Current

Table 1. DC Parameters

Parameters	Description	Test Conditions	7C122 Min.	7C122 Max.	7C148/9 Min.	7C148/9 Max.	7C189/90 Min.	7C189/90 Max.	Units
V_{OH}	Output High Voltage	V_{CC} = Min., $I_{OH} = -5.2$ mA	2.4		2.4		2.4		V
V_{OL}	Output Low Voltage	V_{CC} = Min., I_{OL} = 8.0 mA		0.4		0.4		0.4	V
V_{IH}	Input High Voltage		2.1	V_{CC}	2.0	V_{CC}	2.0	V_{CC}	V
V_{IL}	Input Low Voltage		−3.0	0.8	−3.0	0.8	−3.0	0.8	V
I_{IL}	Input Low Current	V_{CC} = Max., $V_{IN} = V_{SS}$		10		10		10	μA
I_{IH}	Input High Current	V_{CC} = Max., $V_{IN} = V_{CC}$		10		10		10	μA
I_{OFF}	Output Current (High Z)	$V_{OL} < V_{OUT} < V_{OH}$, T_A = Max.	−10	+10	−10	+10	−10	+10	μA
I_{OS}	Output Short Circuit Current	V_{CC} = Max., 0°C < T_A < 70°C		−70		−90		−275	mA
		$V_{OUT} = V_{SS}$, −55°C < T_A < 125°C		−80		−90		−350	mA

When operated in the TTL range, they perform normally. Operated in full CMOS mode, an additional benefit of power savings is realized as the current consumed in the input converter decreases as the input voltage rises above 3.0V, or falls below 1.5V. Since the input signal is in the 1.5 to 3.0V range only when transitioning between logic states, the power savings in a large array with true CMOS inputs can be significant. With input signals on over half of the pins of a device, significant savings in a large system can be realized by using CMOS input voltage swings even in TTL systems.

Switching Characteristics

Although this application note does not directly deal with the AC characteristics of high speed RAMs, the input and output characteristics of these devices have a great deal to do with the actual AC specifications. Conventionally, all AC measurements associated with high speed devices are done at 1.5V and assume a maximum rise and fall time. This eliminates the variations associated with the various configurations that the device will be used in (as a figure of merit when testing the device) but, does not mean that the designer can ignore these influences when designing a system. Maximum rise and fall time is usually found in the notes included on every data sheet. For the products referred to in this application note, a 10 ns maximum rise and fall time is specified for all devices with access times equal to or greater than 25 ns and a 5 ns maximum rise and fall time for all devices with access times less than 25 ns. The AC load and its Thévenin equivalent in *Figure 3* represent the resistive and capacitive components of load which the devices are specified to drive. With either of these loads, the device will be required to source or sink its rated output current at its specified output voltage. The capacitance stresses the ability of the device output to source or sink sufficient current to slew the outputs at a high enough rate to meet the AC specifications. The high impedance load is a convenience to testing when trying to determine how rapidly the output enters a high impedance condition. Once the output enters a high impedance mode, the resistive divider will charge the capacitance until equilibrium is reached. Allowing for noise margin, testing for a 500 mV change is normal. By using a smaller capacitance than normal, the change will occur more quickly, allowing a more accurate determination of entry into the high impedance state.

SWITCHING THRESHOLD VARIATIONS

Switching threshold variations along with input rise and fall times can have an effect on the performance of any device. Input rise and fall times are under the control of the designer, and are primarily affected by capacitive loading, the driver and bus termination techniques. Switching threshold is affected by process variations, changes in V_{CC} and temperature. Compensation of these variables is the territory of the manufacturer, both at the design stage and the manufacturing of the device. Combined threshold shifts over full military temperature ranges and process variations average less than 100 mV. This translates directly to V_{IL} and V_{IH} variations which track well within the noise margins of normal system design particularly since the V_{OL} and V_{OH} changes track to the same 100 mV.

Input Protection Mechanisms

THE ELECTROSTATIC DISCHARGE PHENOMENON

Because of their extremely high input impedance and relatively low (approximately 30V) breakdown voltage, MOS devices have always suffered from destruction caused by ESD (Electro Static Discharge). This has caused two actions. First, major efforts to design input protection circuits without impeding performance has resulted in MOS devices that are now superior to bipolar devices. Second, care in handling semiconductors is now common practice. Interestingly enough, bipolar products that once did not suffer from ESD have now suddenly become sensitive to the phenomenon, primarily because new processing technology involving shallow junctions is in itself sensitive. MOS devices are in many cases now superior to bipolar products. A sampling of competitive BIPOLAR and NMOS 64 bit, 1K bit and 4K bit products reveals breakdown voltages as low as ±150V to greater than ±2001V magnitudes. The circuit in *Figure 4* is used to protect Cypress products against ESD. It consists of two thick oxide field transistors wrapped around an input resistor and a thin oxide device

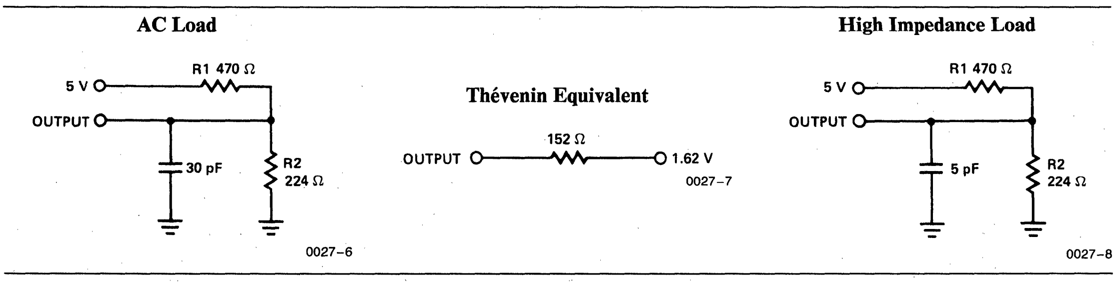


Figure 3. Test Loads

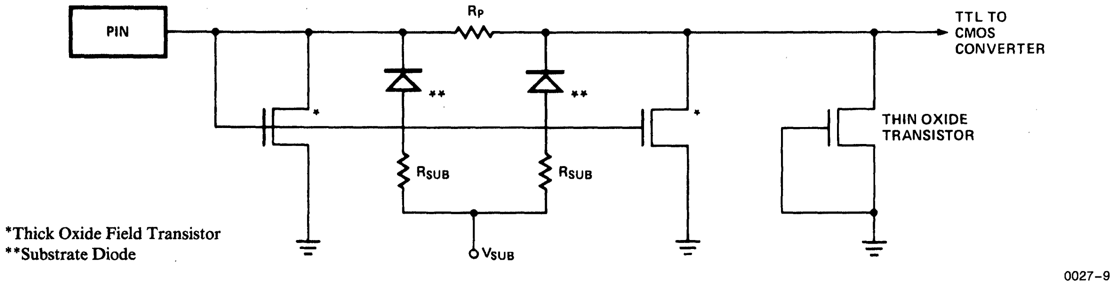


Figure 4. Input Protection Circuit

with a relatively low breakdown voltage of approximately 12V. Large input voltages cause the field transistors to turn on discharging the ESD current harmlessly to ground. The thin oxide transistor breaks down when the voltage across it exceeds the 12V level and it is protected from destruction by the current limiting of R_P. The combination of these two structures provides ESD protection greater than 2250V, the limit of the testing equipment available. In addition, repeated applications of this stress do not cause a degradation that could lead to eventual device failure as observed in functionally equivalent devices.

CMOS Latchup

The parasitic bipolar transistors shown in *Figure 5* result in a built-in silicon controlled rectifier illustrated in *Figure 6*. Under normal circumstances the substrate resistor R_{SUB} is connected to ground. Therefore, whenever the signal on the pin goes below ground by one diode drop, current flows from ground through R_{SUB} forward biasing the lower transistor in the effective SCR. If this current is sufficient to turn on the transistor, the upper PNP transistor is forward biased, the SCR turns on and normally destroys the device. Several solutions are obvious, decreasing the substrate resistance, or adding a substrate bias generator are two. The bias generator technique has several additional benefits, however, such as threshold voltage control which increases device performance and is employed in all Cypress products, along with guard rings which effectively isolate input and output structures from the core of the device and thus effectively decrease the substrate resistance by short circuiting the current paths. Latchup can potentially be induced at either the inputs or outputs. In true CMOS output structures as discussed above, the output driver has a PMOS pullup which creates additional vertical bipolar PNP transistors compounding the latchup problem. Additonal isolation using the guard ring technique can be used to solve this problem, at the expense of additional silicon

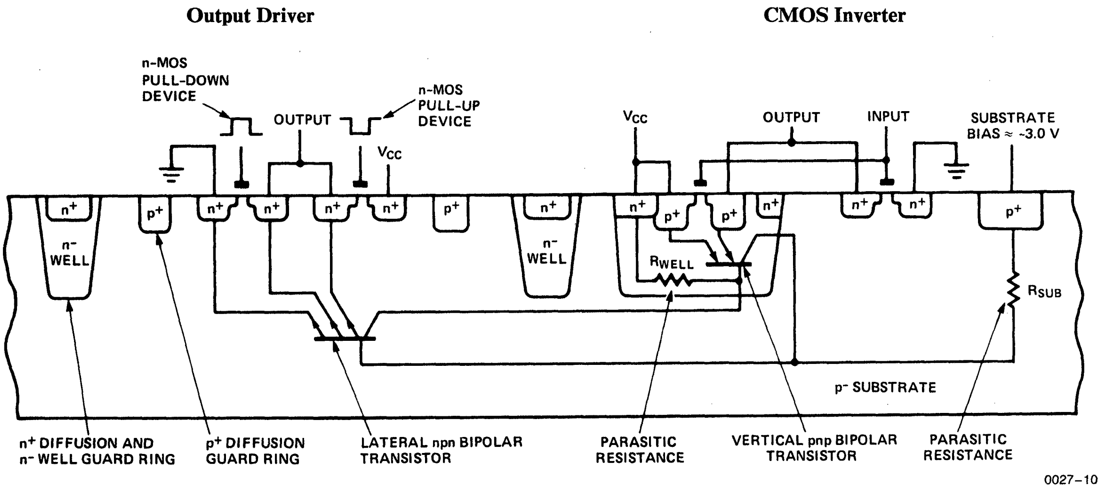


Figure 5. CMOS Cross Section and Parasitic Circuits

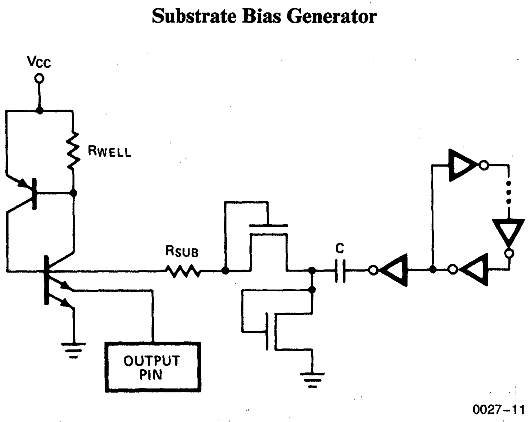


Figure 6. Parasitic SCR and Bias Generator

area. Since all of the devices of concern here require TTL outputs, the problem is totally eliminated through the use of an NMOS pullup.

LATCHUP CHARACTERISTICS

Inducing Latchup for Testing Purposes

Care needs to be exercised in testing for latchup since it is normally a destructive phenomena. The normal method is to power the device under test with a supply that can be current limited, such that when latchup is induced, insufficient current exists to destroy the device. Once this setup exists, driving the inputs or outputs with a current, and measuring the point at which the power supply collapses will allow non-destructive measurement of the latchup characteristics of the devices under question. In actual testing, with the device under power, individual inputs and outputs are driven positive and negative with a voltage and the current measured at which the device latches up. This provides the DC latchup data for each pin on the device as a function of trigger current.

Measurement of Latchup Susceptibility

Actually measuring the latchup characteristics of devices should encompass ranges of reasonable positive and negative currents for trigger sources. Depending on the device, latchup can occur as low as a few mA to as high as several hundred mA of sink or source current. Devices which latch at trigger currents of less than 20 to 30 mA are in danger of encountering system conditions that will cause latchup failure.

Competitive Devices

Although there are few devices directly competitive with the Cypress devices covered in this application note, the latchup characteristics of the closest functionally similar devices were measured. The results show devices that latchup at as low as 10 mA all the way to devices that can sustain greater than 100 mA of trigger current without latchup. The Cypress devices covered in this document can sustain greater than 200 mA without incurring latchup, far more than is possible to encounter in any reasonable system environment.

Elimination of Latchup in Cypress RAMs

Since the latchup characteristic is one that inherently exists in any CMOS device, rather than change the laws of physics, we design to minimize its effects over the operating environment that the device must endure. These include temperature, power supply and signal levels as well as process variations. There are several techniques employed to eliminate the latchup phenomenon. Two of them involve moving the trigger threshold outside the operating range as to make it impossible to ever encounter it. These are either using low impedance, epitaxial, substrates and/or a substrate bias generator. The use of a low impedance substrate has the effect of increasing the undershoot voltage required to generate the required trigger current that causes latchup. A substrate bias generator has two effects which help to eliminate latchup. First, by biasing the substrate at a negative, −3.0V, voltage, the parasitic diodes can not be forward biased unless the undershoot exceeds the −3V by at least one diode drop. Second, if undershoot is this severe, the impedance of the bias generator itself is sufficient to deter sufficent trigger from being generated. The bias generator has one additional noticeable characteristic, it effectively removes the input clamp diode. This is due to the anode of the diode connecting to the substrate which is at −3.0V. Therefore, even though the diode exists as shown in *Figure 4*, DC signals of −3.0V do not forward bias the diode and exhibit the clamp condition. The benefits of this are apparent in higher noise tolerance as substrate currents due to input undershoot do not occur.

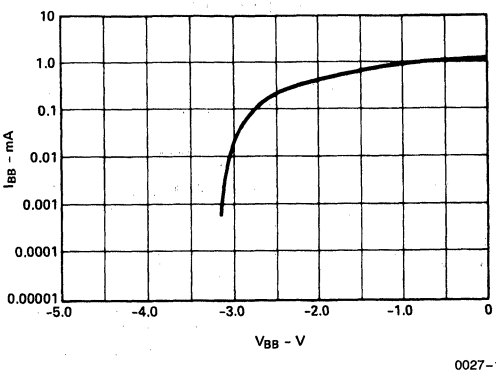
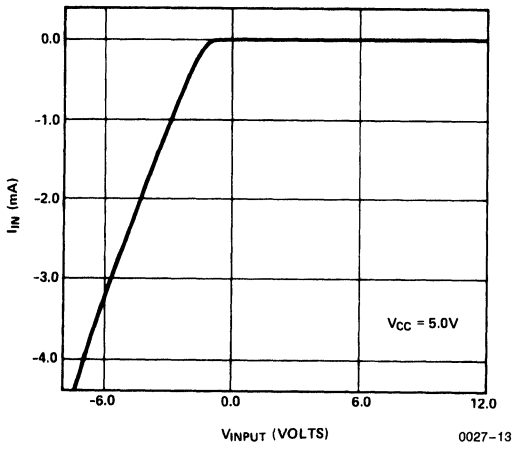
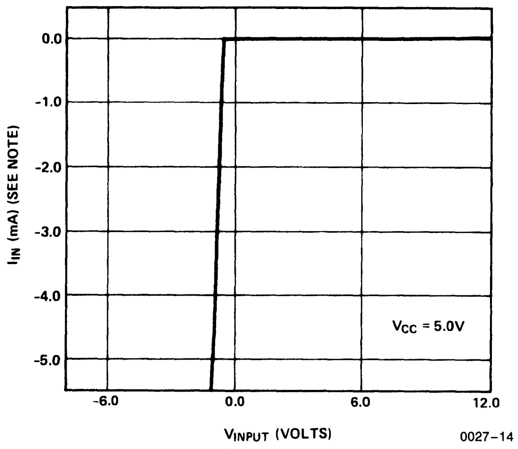


Figure 7. Bias Generator Characteristics





Note: Output is in a High Impedance Condition.

Figure 8. Input V/I Characteristics

Figure 9. Output V/I Characteristics

Figures 8 and *9* represent the voltage and current characteristics of the devices discussed in this application brief. *Figure 8* is characteristic of an input pin, and *Figure 9* an output pin in a high impedance state. In *Figure 8,* the input covers + 12V to − 6V, well outside the + 7V to − 3V specification. Referring to *Figure 4* to understand these characteristics, when the input voltage goes negative, the thin oxide transistor acts as a forward biased diode and the slope of the curve is set by the value of R_P. As the input voltage goes positive, only leakage current flows. The output characteristics in *Figure 9* show the same phenomenon, with the exception that, since this is not an input, no protection circuit exists, and therefore no R_P exists. An equivalent thin film device acts as a clamp diode which limits the output voltage to approximately − 1V at − 5 mA.

NOTES:

CYPRESS SEMICONDUCTOR

74F189 Application Brief

Introduction

There are available in the market a number of high speed 64 bit static RAMs organized 16 by 4 bits. Because of the various different manufacturers specifications, there is no apparent true second source for these products as each operates with some unique characteristics. The composite specifications contained in this applications brief will allow the interchangeable use of the Cypress CY7C189 with the 74F189 and the Cypress CY7C190 with the 74F219 with optimization for either power or performance.

Specifications

Depending on system requirements, the SPEED OPTIMIZED specification will allow the designer to select performance at the expense of power, and use either Cypress's CY7C189-15 or the 74F189 interchangeably. If, however, the major criteria is power the designer can achieve a 55 mA max power specification using the Cypress CY7C189-25 interchangeably with the 74F189 by designing with the POWER OPTIMIZED specification.

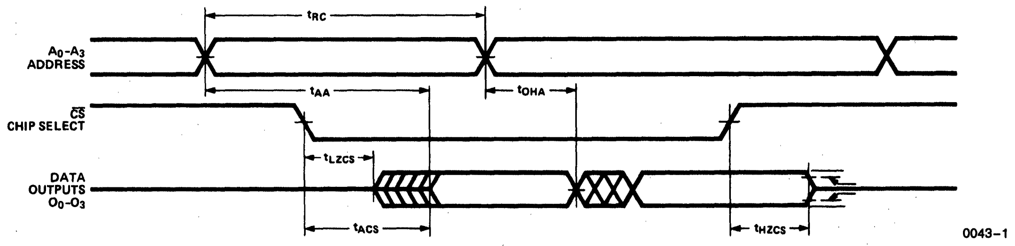
Electrical Characteristics Over the Operating Range

Parameters	Description	Test Conditions		Speed Optimized		Power Optimized		Units
				Min.	Max.	Min.	Max.	
V_{OH}	Output HIGH Voltage	V_{CC} = Min., I_{OH} = −3.0 mA		2.4		2.4		V
V_{OL}	Output LOW Voltage	V_{CC} = Min., I_{OL} = 16.0 mA			0.5		0.5	V
V_{IH}	Input HIGH Voltage			2.0	V_{CC}	2.0	V_{CC}	V
V_{IL}	Input LOW Voltage			−3.0	0.8	−3.0	0.8	V
I_{IX}	Input Leakage Current	GND ≤ V_I ≤ V_{CC}		−600	+20	−600	+20	μA
I_{OZ}	Output Leakage Current	GND ≤ V_O ≤ V_{CC}		−50	+50	−50	+50	μA
I_{OS}	Output Short Circuit Current	V_{CC} = Max., V_{OUT} = GND			−150		−150	mA
I_{CC}	Power Supply Current	V_{CC} = Max., I_{OUT} = 0 mA	Commercial		90		55	mA
			Military				70	mA

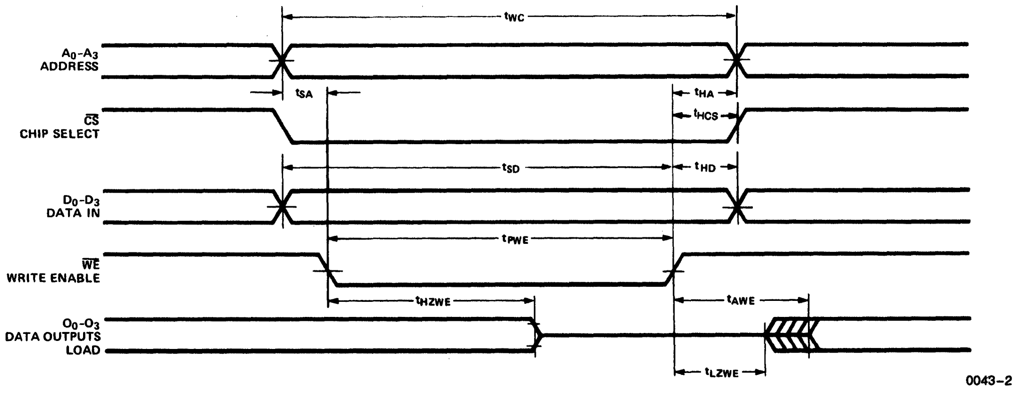
Switching Characteristics Over the Operating Range

Parameters	Description	Speed Optimized		Power Optimized		Units
		Min.	Max.	Min.	Max.	
READ CYCLE						
t_{RC}	Read Cycle Time	27		27		ns
t_{ACS}	Chip Select to Output Valid		14		15	ns
t_{HZCS}	Chip Select Inactive to High Z		12		15	ns
t_{LZCS}	Chip Select Active to Low Z		12		15	ns
t_{OHA}	Output Hold from Address Change	5		5		ns
t_{AA}	Address Access Time		27		27	ns
WRITE CYCLE						
t_{WC}	Write Cycle Time	15		20		ns
t_{HZWE}	Write Enable Active to High Z		14		20	ns
t_{LZWE}	Write Enable Inactive to Low Z		12		20	
t_{AWE}	Write Enable to Output Valid		29		29	ns
t_{PWE}	Write Enable Pulse Width	15		20		ns
t_{SD}	Data Setup to Write End	15		20		ns
t_{HD}	Data Hold from Write End	0		0		ns
t_{SA}	Address Setup to Write Start	0		0		ns
t_{HA}	Address Hold from Write End	0		0		ns
t_{HCS}	Chip Select Hold from Write End	6		6		ns

Read Cycle



Write Cycle



Section Contents

CYPRESS SEMICONDUCTOR

Introduction to Diagnostic PROMs

Scope and Purpose

This Application Brief will provide the reader with a basic understanding of the concept of a diagnostic PROM, as well as a brief introduction to possible applications.

Beginning with a short tutorial on system diagnostics, the reason for incorporating diagnostics into a design and the special testability problems associated with sequential designs are presented. The concept of shadow-register-based diagnostics is presented, and the benefits of this approach are outlined.

Next, a description of Diagnostic PROMs is given. This covers the similarity/dissimilarity of diagnostic PROMs relative to standard registered PROMs, as well as fundamental operation of a diagnostic PROM followed by a description of the Cypress CY7C268 and CY7C269 8K x 8 Diagnostic PROMs.

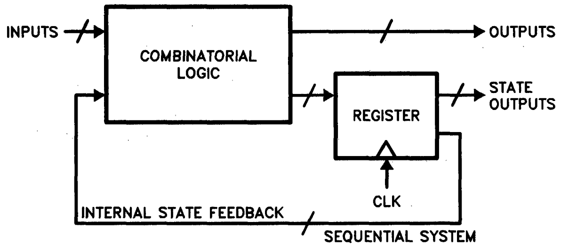
In conclusion, an application example is presented.

Introduction to System Diagnostics

As electronic systems continue to grow in size, functionality, and complexity, it is becoming increasingly difficult to test them and determine their reliability, as well as to service the end product in the field. One way to simplify the task of testing electronic systems is to design some form of testability into the system.

Controllability and observability are the key points of testability. These two qualities are easily obtained for a combinatorial system where the outputs are strictly a function of the current inputs. Test vector methods are easily devised and implemented for combinatorial systems. But, for a sequential system, where the outputs are a function of both the current inputs and the previous state(s), controllability and observability may be lost due to lack of access to the internal states of the machine. Consequently, building testability into a system means being able to control and observe all possible states of a system.

Consider the simple sequential machine in *Figure 1.* As is evident, access to internal states—which is necessary for complete controllability and observability—is either denied or difficult to obtain. The obvious way to add testability to this system is to permit access to these internal states. One way to gain this access is through addition of a diagnostic shadow register, as shown in *Figure 2.* Observability is effected by adding a serial data output path (SDO) to allow shifting internal state information out of the system. Controllability is gained by permitting a serial data input path (SDI) to set the state of the internal registers. As a result, relatively simple test vector methods can again be used to test the system. Consider, for example, the complex sequential machine shown in *Figure 3.* This system would be virtually impossible to test in the current configuration, due to the fact that we can not control or observe the internal states of the machine. In order to increase the testability of this machine, observability must be added at points 01, 02, and 03. If this were accomplished, one would be able to observe the internal states of the machine. Additionally, controllability must be added at points C1, C2, and C3. This would enable the internal states of the machine to set. This controllability and observability can be attained by adding shadow registers, as depicted in *Figure 4.* The result is a complex sequential machine with a



0125-1

Figure 1. Simple Sequential Machine

February 1988

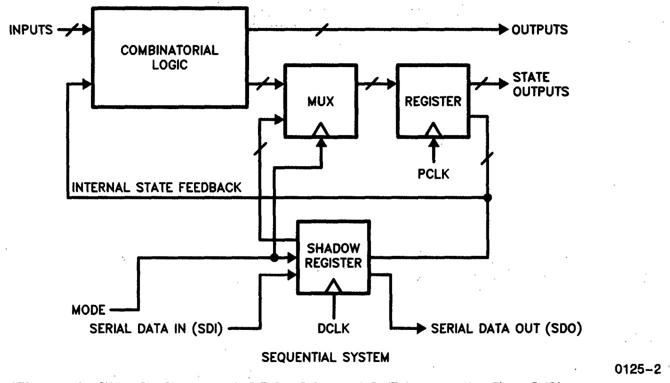


Figure 2. Simple Sequential Machine with Diagnostic Capability

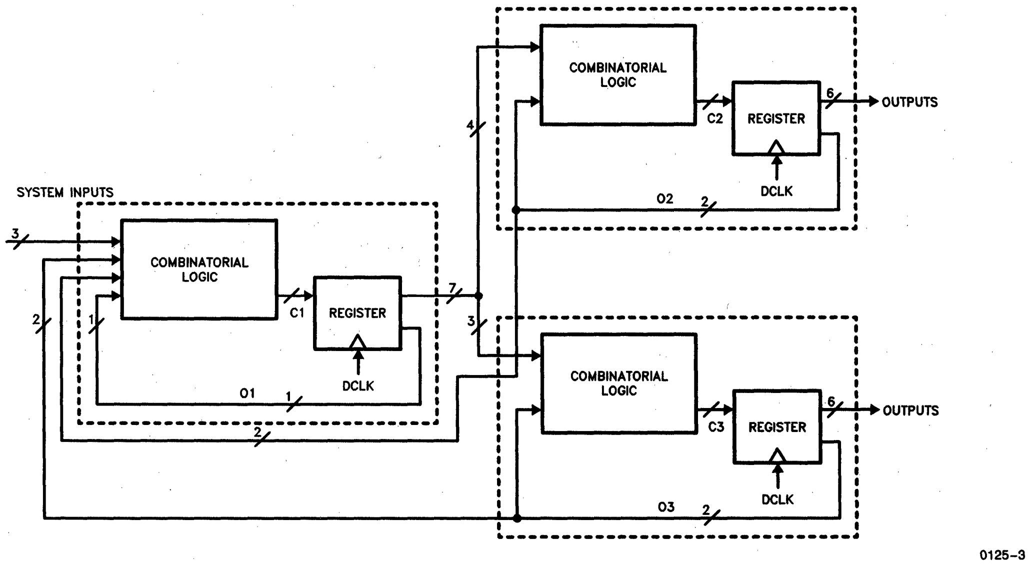


Figure 3. Complex Sequential Machine

high degree of testability. As a result of these actions, simple test vector methods can now be used to fully test the machine. For instance, the state of the register at point C1 can be set, the machine may be clocked through some known number of cycles, and the state of the machine may be observed at points 01, 02, and 03.

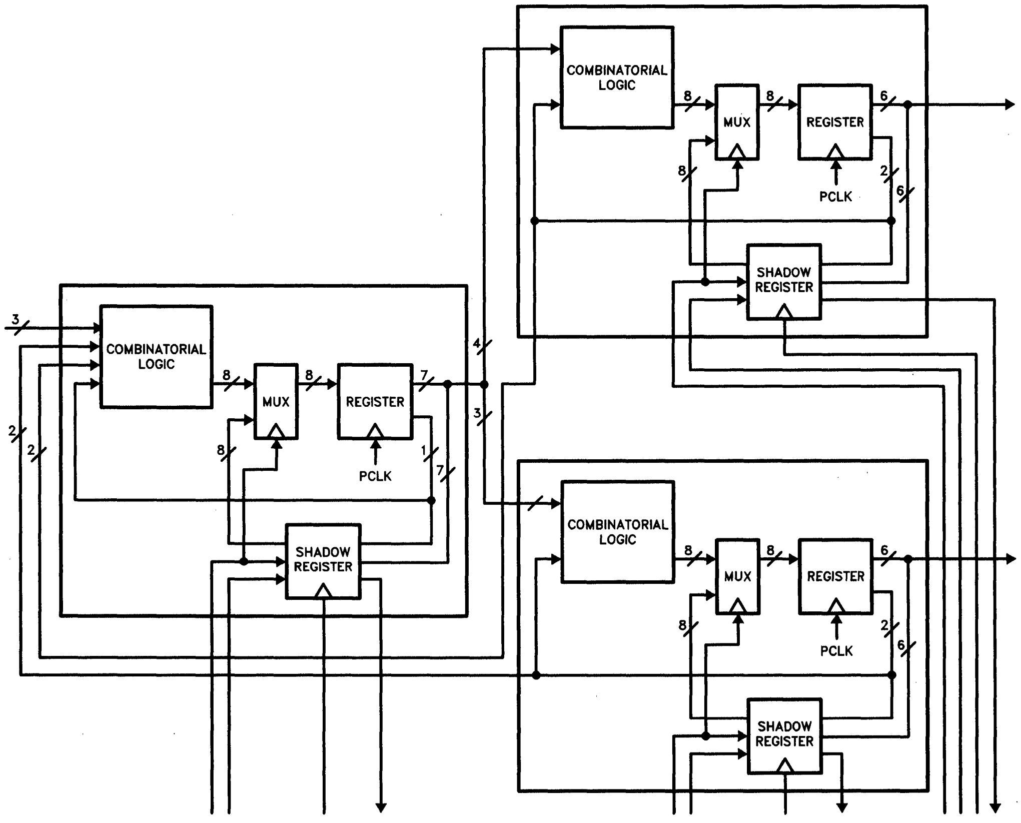
Knowing what state the machine should be in at that particular point in time at each observation point, the known "correct" state of the machine can be compared with the observed machine state (at each observation point), thereby determining if: a) the machine is functioning correctly; and b) if not, which "machine primitive" is not functioning correctly (fault detection). Note that this approach to sequential design will also permit testing to see what the machine would do if a glitch caused a jump into an unused state, which in turn makes the design task of forcing the machine back into a known state much less complex.

The real advantage of this approach is that it requires no changes in architecture, minimal hardware changes, and results in a minimal (5–10%) area hit when integrated into existing integrated circuits.

Diagnostic PROMs

Diagnostic PROMs are a relatively minor migration from standard registered PROMs. A block diagram of a diagnostic PROM is presented in *Figure 5*. The addition of diagnostic capability to a registered PROM includes the addition of:

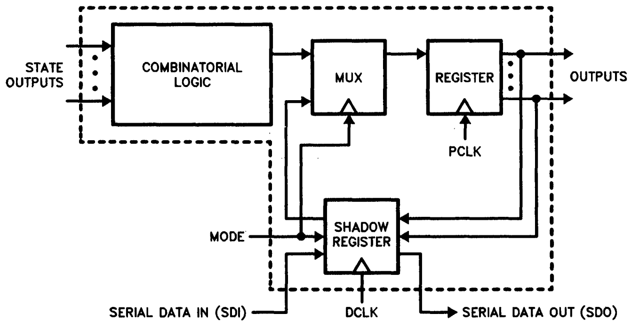
—a shadow register
—multiplexer
—MODE pin
—SDI (Serial Data In) pin
—SDO (Serial Data Out) pin
—diagnostic clock



0125-4

Mode, SDI, SDO, and DCLK for each "Machine Primitive"

Figure 4. Complex Sequential Machine with Diagnostic Capability



0125-5

Figure 5. Diagnostic PROM Block Diagram

The shadow register is dynamically configured based on the value of the mode signal. If mode is set such that the user desires to input data to the PROM, the shadow register is configured as serial-in, parallel-out; if the user desires to extract information from the PROM, the shadow register is configured as a parallel-in, serial-out. So the shadow register serves two purposes: First, the shadow register can be configured to serially receive the data that can be transferred to the register containing state information and appear at the outputs during the next cycle. The obvious advantage of this feature is that it allows the user to effectively preset the condition that will be sent through the part of the system that "follows" the PROM; ie, the user can insert state information into the system. This feature adds controllability to the system.

The second purpose that the shadow register serves is to allow the user transfer data from the register containing state information and to serially shift that data out of the PROM. This feature adds observability by allowing the user to observe the state of the PROMs pipeline register at any given point in time. Inclusion of the above named features in a registered PROM can therefore add testability to any system by providing the user with the mechanism to build both controllability and observability into his system. Note that this increase in functionality is effected without loss of other desirable registered PROM features such as programmable initialization, programmable output enable, etc.

Cypress Diagnostic PROMs

Cypress Semiconductor manufactures two Diagnostic PROMs, the CY7C268 and CY7C269. These 64K byte-wide Diagnostic PROMs are manufactured in CMOS for the optimum speed/power tradeoff resulting in 550 mW power dissipation while maintaining 40 ns maximum set-up and 20 ns clock-to-output. Both contain an edge-triggered pipeline register and on-chip diagnostic shift register. Both are capable of withstanding >2001V ESD. Both are produced in our EPROM-based process, which allows testing for 100% programmability. Both are available in PLCC/LCC and Dual Inline Packages, and both are available in a windowed package for reprogrammability. The CY7C268 features full diagnostic capability and is available in 32-lead PLCC/LCC or 32-pin 0.5 in DIPs. The CY7C269 features limited diagnostic capability and is available in 28-lead PLCC/LCC or 28-pin 0.3 in DIPs.

For an in-depth description of functionality, refer to the data sheet. The following discussion briefly describes the diagnostic functions available in each device.

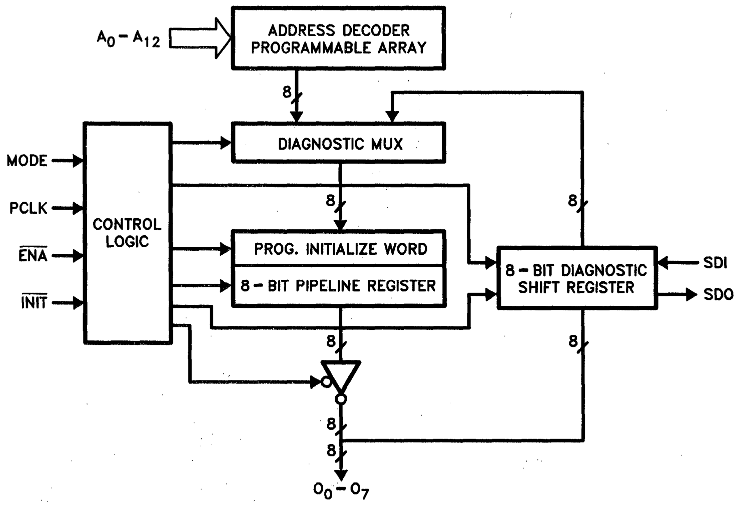
CY7C268

A condensed block diagram of the CY7C268 is presented in *Figure 6.* The pin names and functions of the CY7C268 are as follows:

Name	I/O	Function
A_0–A_{12}	I	Address Input
O_0–O_7	O	Data Lines
$\overline{ENA}$	I	Synchronous or Asynchronous Output Enable
$\overline{INIT}$	I	Asynchronous Initialize
MODE	I	Sets PROM to Operate in Pipelined or Diagnostic Mode
DCLK	I	Diagnostic Clock (Used to Clock the Shadow Register)
PCLK	I	Pipeline Clock (Used to Clock the Output Registers)
SDI	I	Serial Data In (Used to Serially Shift Data into the Diagnostic Register)
SDO	O	Serial Data Out (Used to Serially Shift Data Out of the Diagnostic Register)

Note that full diagnostic capability is realized through the use of four control signals: **SDI** (Serial Data In), **SDO** (Serial Data Out), **MODE,** and **DCLK** (diagnostic clock). Inclusion of both DCLK and PCLK assures that serial data can be shifted into or out of the diagnostic register while the PROM is operating in normal pipeline fashion. As a result, the CY7C268 has three possible modes of operation:

i. normal (pipelined)
ii. diagnostic
iii. both simultaneously



A0 - A12
ADDRESS DECODER PROGRAMMABLE ARRAY
8
DIAGNOSTIC MUX
MODE
PCLK
ENA
INIT
CONTROL LOGIC
PROG. INITIALIZE WORD
8 - BIT PIPELINE REGISTER
8 - BIT DIAGNOSTIC SHIFT REGISTER
SDI
SDO
8
O0 - O7
0125-6

Figure 6. Condensed Block Diagram of the CY7C268

The following table summarizes the operational modes of the CY7C268:

Data Flow Description	Mode	$\overline{ENA}$[1]	SDI	SDO	DCLK	PCLK
Normal Operation[1]	L	H, L	DATA IN	SDO	—	↑
Shadow to Pipeline[1]	H	H, L	X	SDI	—	↑
Pipeline to Shadow	H	L	L	SDI	↑	—
Data In to Shadow	H	H	L	SDI	↑	—
Shift Shadow Reg.[1]	L	H, L	DATA IN	SDI	↑	—
No Operation[1]	H	H, L	H	SDI	↑	—

Note:
1. For the asynchronous enable operation, data out is enabled on the first LOW to HIGH clock transition after $\overline{E}$ is brought LOW. When $\overline{E}$ goes from LOW to HIGH (enable to disable) the outputs will go to the high impedance state (after a propagation delay) immediately if the asynchronous enable was programmed. If the synchronous enable was selected, a LOW to HIGH transition is required.

CY7C269

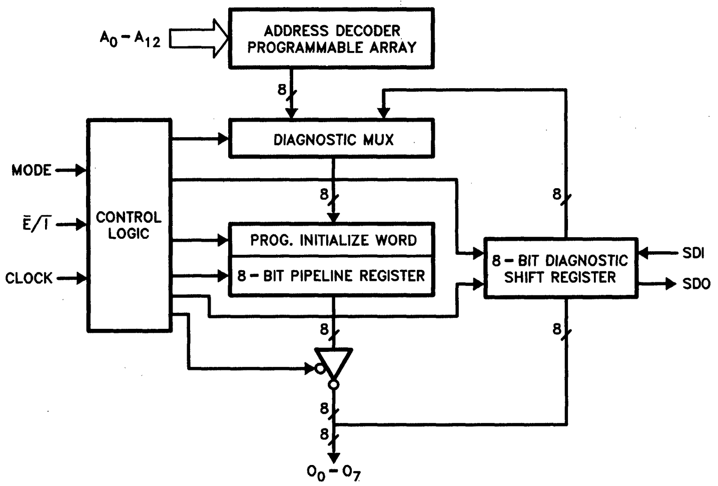
A condensed block diagram of the CY7C269 is presented in *Figure 7.* As is evident, the CY7C269 has reduced diagnostic functionality relative to the CY7C268. The CY7C269 is ideal for applications requiring limited diagnostics with a premium on board space conservation, and is available in 28-pin, 300 mil DIPs (windowed or opaque) and in 28-lead PLCC/LCC packages. The pin names and functions of the CY7C269 are as follows:

Name	I/O	Function
A_0–A_{12}	I	Address Inputs
O_0–O_7	O	Data Lines
$\overline{E}$, $\overline{I}$	I	Enable or Initialize
Clock	I	Pipeline and Diagnostic Clock
Mode	I	Sets PROM to operate in either diagnostic or regular pipelined mode (note that the two modes are mutually exclusive).
SDI	I	Serial Data In
SDO	O	Serial Data Out

Note that limited diagnostic capability is realized through inclusion of three diagnostic signals: **MODE, SDI,** and **SDO**. Since there is only one **CLOCK**, the regular and diagnostic modes are mutually exclusive. The following table summarizes the operating modes of the CY7C269:

Data Flow Description	Mode	$\overline{E}$, $\overline{I}$	Clock	SDI	SDO
Normal Operation	L	[1][2]	↑	X	High Z
Shadow to Pipeline	H	L	↑	L	SDI
Pipe or Bus to Shadow	H	L	↑	H	SDI
Shift Shadow	H	H	↑	Data In	SDO

Notes:
1. $\overline{E}$ or $\overline{I}$ function selected during programming.
2. If I selected, outputs always enabled. If $\overline{E}$ selected, outputs are enabled synchronously or asynchronously as programmed.
3. If $\overline{I}$ selected, outputs always enabled. If $\overline{E}$ selected, during diagnostic operation the data outputs will remain in the state they were in when the mode was entered. When enabled, the data outputs will reflect the outputs of the pipeline register. Any changes in the data in the pipeline register will appear on the output pins.


A0 - A12
ADDRESS DECODER PROGRAMMABLE ARRAY
8
DIAGNOSTIC MUX
MODE
E/I
CLOCK
CONTROL LOGIC
PROG. INITIALIZE WORD
8 - BIT PIPELINE REGISTER
8 - BIT DIAGNOSTIC SHIFT REGISTER
SDI
SDO
O0 - O7

0125-7

Figure 7. Condensed Block Diagram of the CY7C269

Design Example

As an example, consider the complex sequential machine presented earlier. This machine could be easily implemented using CY7C268's or CY7C269's, as shown in *Figure 8.* Note that the block labeled "diagnostic control" could consist of PLDs, PROMs a sequencer, or a small microcontroller. The choice between using the CY7C268 or the CY7C269 would be based complexity of the diagnostic function required. For full diagnostics that can function simultaneously with regular pipelined operation, the CY7C268 should be used. For an application where limited diagnostic capability is required—perhaps only a power-up or at some other well-defined point in time—the CY7C269 may be used.

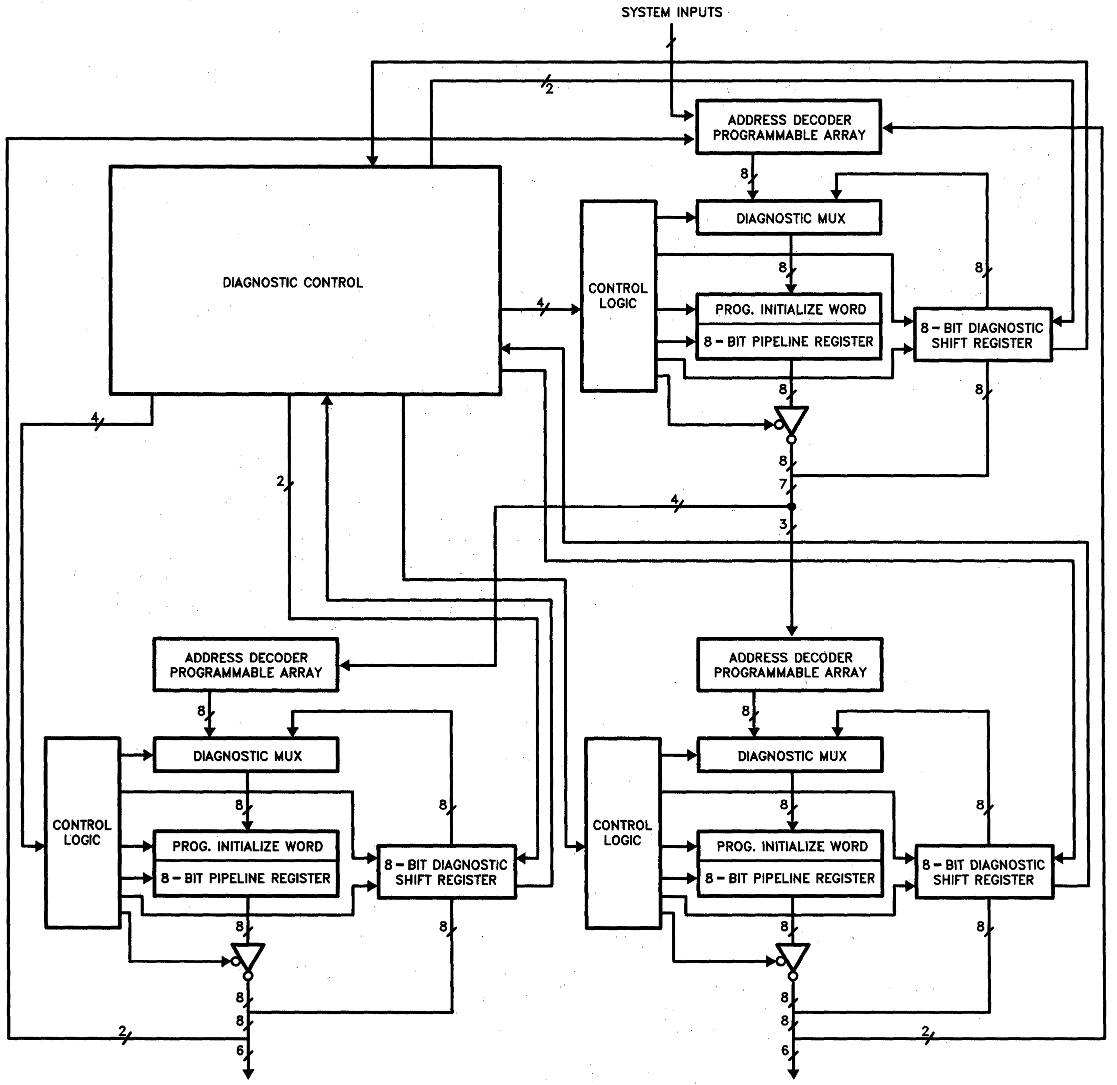

SYSTEM INPUTS
ADDRESS DECODER PROGRAMMABLE ARRAY
DIAGNOSTIC MUX
PROG. INITIALIZE WORD
8 - BIT PIPELINE REGISTER
8 - BIT DIAGNOSTIC SHIFT REGISTER
CONTROL LOGIC
DIAGNOSTIC CONTROL
8
4
2
7
3
6
0125–8

Figure 8. Complex Sequential Machine Implemented with Cypress Diagnostic PROMs

CYPRESS SEMICONDUCTOR

Pin-Out Compatibility Considerations of SRAMs and PROMs

When looking for pin compatible replacements for PROMs, there are a number of key parameters that must be met. This application brief discusses the non-electrical parameters of pin-out and programming involved in finding socket compatible second sources for PROMs. Comparison with the selection of a socket compatible SRAM second source is provided. Additionally, an example of a verified conversion from the Motorola 68764 to the Cypress CY7C264, a PROM conversion that is not address line compatible, is presented.

Ignoring the AC/DC characteristics, finding a second source for an SRAM is relatively simple. As long as the power, ground, control (chip select, read, write), address, and data lines are on the same pins the devices should be compatible. **Specifically, on SRAMs, the address and data lines need not be numbered identically** between the two devices being compared for them to function identically in the same socket. As an example, on several Cypress SRAMs, the address pin numbering is not the same as some of our competitors. Let's look at a simplified example that illustrates why this is not a problem. Let's assume that we have a new device, the 2 bit x 4 location SRAM:

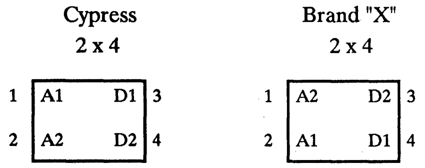


Figure 1. Example 2x4 Simplified SRAMs

Note that the inferior pin out chosen by the Brand "X" 2 x 4 assigns Address line 2 (A2) to pin 1 whereas the superior pin-out used by the Cypress device has A1 at pin 1, etc. It is our assertion that these simplified devices are pin compatible. Let's assume that our engineering staff designed an infra-red scanning pattern recognizing toaster oven with the Brand "X" data sheet. Just as your company is about to ramp into volume production, Brand "X" sends out an End Of Life notice on their 2 x 4, because they are converting all of their capacity to making DRAM memories. At this point, you have no desire to layout a new P.C. board, so let's take a look at how these devices would look in your design.

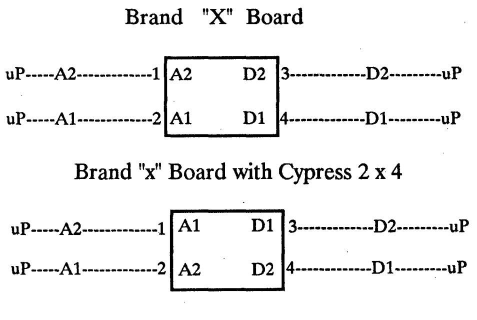

Brand "X" Board
uP-----A2------------1 A2 D2 3------------D2---------uP
uP-----A1------------2 A1 D1 4------------D1---------uP
Brand "x" Board with Cypress 2 x 4
uP-----A2------------1 A1 D1 3------------D2---------uP
uP-----A1------------2 A2 D2 4------------D1---------uP

Figure 2. Example System with 2x4 SRAMs

In this case, uP is a microprocessor interfacing to the SRAM. What is of key importance is that the data read from a given address generated by the microprocessor is the same as data written to the same location earlier. With a SRAM, any inconsistency between the Address and Data line numbering does not really matter because the data read will be the same as the data previously written. This occasionally causes some concern with customers who have not seen this before. To illustrate our point suppose that we write a value of 1 (uP:D2,D1 = 0,1) at location 2 (uP:A2,A1 = 1,0). If we read location 3, we will obtain the value 1 that was written, because the address presented to the SRAM during the read is the same as the address for the previous write. Similarly the data read will be in the same bit order as presented during the previous write to the location. Thus so far as our system is concerned, the two SRAM devices are compatible. The only difference, that is not significant to our system, is where inside the SRAM the data was actually stored. In the Cypress device, the uP address of 2 (uP:A2,A1 = 1,0) actually stored the data at SRAM location 1 (Cypress:A2,A1 = 0,1). In the Brand "X" RAM, the data is physically stored in location 2. However the address translation is transparent to the uP. Since the same location is accessed for the subsequent reads, the difference in address numbering between the two devices doesn't really matter to our system. Similarly, any numbering difference on the data lines doesn't matter either. The point that is of primary importance here is that for SRAMs, all writes and reads are generated in your system, and thus so long as the address and data lines are on the same pins, differences in the numbering doesn't matter.

For PROMS, the scenario becomes slightly more complex. Since PROMS are programmed using a programmer that is separate from the system in which they are used, it becomes more difficult to substitute a PROM with a device that does not have the same address and/or data pin numbering. Let's assume that our Hi-Tek toaster oven's 2 x 4's are now PROMS. If we programmed each location with data, we would find that the Cypress device would not work properly when used in the Brand "X" designed socket. In this case our programmer put the data at location 2, and the board would read this data when the microprocessor requested the data at location 3. Additionally the data bits will have been swapped on this read. What a mess! It becomes apparent that it is easiest to replace this PROM with a device that has the same address and data line numbering. There are still methods that we can use that will allow us to use the Cypress 2 x 4 PROM in this socket that we will consider.

The objective in trying to make the Cypress PROM work in the foreign pin-out socket is to have the data read by the system be the same as the data read when the Brand "X" device is used. In our 2 x 4 example, there are two problems - address line numbering mismatch and Data line numbering mismatch. Let's first address the data line mismatch. As it stands data that was written in as bit1,bit2 is read as bit2,bit1 or swapped. If we were able to change our PC Board layout, we could fix this problem by swapping the printed traces for D1 and D2. Unfortunately this would also disallow the use of the Brand "X" device on our board. If we could internally swap the data bits in the Cypress device, then when they were read they would be in the correct order. This swapping of the data bits in the Cypress device can be achieved through several means. First, we might modify our programming adapter such that D2 and D1 are swapped from the normal order when programming the part. Then when the device is read, we would get the bits in the same order as presented by the Brand "X" device. This is not a recommended method of solving the problem, because modifying programmers tends to make the manufacturer of the programmer unhappy. A second method of solving this problem is to alter the binary image of the PROM contents such that bits D1 and D2 are swapped in a file on your computers disk, then using this altered binary image file to program the Cypress PROM. This is less likely to cause damage than modifying a programmer, but requires some skill in altering the binary file. Finally, the easiest solution to this problem is to trick the PROM programmer into swapping the bits for you. If you set your programmer for the Cypress device type, read a programmed Brand "X" device into memory, then program the Cypress part with the image in programmer memory, the bits will have been swapped for you. Let's look at how this works.

1) Brand "X" 2 x 4	: Bit 2, Bit 1
2) Programmer (Cypress)	: Bit 1, Bit 2
3) Cypress 2 x 4	: Bit 1, Bit 2
4) System Board uP	: Bit 2, Bit 1

Figure 3. PROM Bit Swapping with Programmer

From the diagram above, we can see that the bits in the Brand "X" device are stored in the order Bit2,Bit1. This is the same order that the uP will read them on our board. When we set the programmer to read the Cypress part, the data lines are logically swapped from the Brand "X" ordering. Thus when we read the Brand "X" part, the data bits will be swapped as shown. When the Brand "X" part is removed from the socket, and the Cypress device is plugged in and programmed, the bits will be programmed into the Cypress part in this same 'reversed' order. When we place the Cypress part into our board, the bits will be swapped again due to the difference in numbering between the Cypress part and the board layout, and the uP will get the data in the correct order.

The second problem that exists is the difference in address line numbering. This problem can be resolved in exactly the same manner as the data swap problem. By simply setting the programmer to the Cypress device type, reading the Brand "X" part, then programming the Cypress part, any addressing differences will be solved allowing the use of the Cypress device. The difference here is that the location of data words will be swapped to allow for the difference in pin-outs, just as the bits were swapped in the data line mismatch case.

Many programmers will allow you to read a device different than the part selected, complaining only during a program if the device types do not match. With such a programmer, carrying out the above procedures to convert a prom should not present a problem. However there are some programmers that will not allow the user to read a device if it is different from the part selected. These programmers will prevent our method from working. Fortunately, the Cypress' CY3000 QuickPro programmer will allow this approach to solving our problem. Cypress Field Applications Engineers, Sales Offices and Distributors can use their QuickPro to generate a Cypress master prom that can be used as a source for copying with un-cooperative programmers.

As an example of such a conversion, the Motorola 68764 8K x 8 prom has a similar pin-out to the Cypress CY7C264 with the exception of address lines 10, 11, and 12.

PIN	Cypress 7C264	Motorola 68764
21	A10	A12
19	A11	A10
18	A12	A11

Figure 4. Cypress 7C264 vs. Motorola 68764 Pin-out

The following procedure will program a Cypress CY7C264 such that it will work properly in a socket designed to accept the Motorola device.

1) Invoke the Cypress QuickPro (or other usable programmer) and select the Cypress 7C264 as the device to be programmed.

2) Place the Motorola part in the programmer adapter socket and read the device. Optionally write the device contents to a disk file.

3) Place a Cypress CY7C264 into the programmer adapter socket and program the part. Optionally the contents of the disk file may be read as the source for programming.

The programmed device will now work in the Motorola designed socket.

Summary

If the pins used for power, ground, control, address, and data line numbering is the same for two devices, they may be used in the same socket if the other electrical parameters are compatible. Differences in Address and Data line numbering are of no consequence in SRAM use. Differences in Address and Data line numbering in PROM device can be compensated for by using a simple programming procedure.

NOTES:

Section Contents

CYPRESS SEMICONDUCTOR

Introduction to Programmable Logic

Why Use a PLD?

One of the fastest growing segments of the semiconductor market today is ASIC (Application Specific Integrated Circuit) devices. ASICs are generally used to integrate SSI/MSI logic chips or functions, thus increasing packaging density and reducing board real estate. Other benefits to the user are reduced power, higher reliability, and product secrecy.

ASICs include several different kinds of devices. There are full-custom devices, standard cells, gate arrays, and PLDs. Full-custom devices offer the greatest degree of integration, but they are expensive and the development cycles can be on the order of nine months to a year. Full-custom designs are justified only for very large volume applications. Standard cell devices can be turned around much more quickly (about four months) and they cost less. However, the level of integration and, thus the speed are less than with the full-custom product. Gate arrays offer even less dense integration, but since only two metal masks must be fabricated, the design turnaround can be as low as six weeks. One drawback of all these ASICs is that the design logic must be set at the start of this cycle; and if it changes, the whole product cycle must start over from scratch. In addition, each device is application specific, so inventory must be watched very carefully to make sure that just enough of each device is ordered to meet demand.

An alternative to custom or semicustom devices is the PLD (Programmable Logic Device). Although PLDs do not offer the same level of integration as the other ASICs, the reduction in board space is still significant. The reduction factor is application dependent. It can be between 4:1 and 10:1 for the smaller PLDs (20 to 24 pins) and 75:1 for high-density/pin-count devices such as the LCA or MAX families. Additional benefits to the user are reduced parts inventory, faster design, and turnaround time, and simplified timing considerations.

Since a PLD is a sold as a "generic" array of logic, customized by the user, the same PLD can be used in many different applications, spanning any number of projects. Cypress's PLDs are based on CMOS EPROM technology, thus making them EPLDs that are erasable using an ultraviolet light source. Design changes can be made at any time of the product cycle more easily than with other ASICs. The design cycle of a PLD of moderate complexity can be a week or less, and after the one-time purchase of a good development software package and programmer, the parts are relatively inexpensive. Timing is simplified, since all logical functions will take approximately the same path through the device. Thus the same propagation delays apply to all outputs of the device. The reasons for this will become clear later.

PLD Technology

All of the Cypress EPLD families (with the exception of the CY7C360 family) are based on the familiar "sum-of-products" architecture. Boolean transfer functions of this form can be implemented by programming the AND array whose output terms feed a fixed OR array. This scheme can implement most combinatorial logic functions and is limited only by the number of product terms available in the AND-OR array. A variety of different sizes and additional architectural features (i.e., flip-flops) are available.

The original TTL PLDs used a fuse as their programmable element. In an unprogrammed device, all of the connections between input pins and product terms were "fused" during the manufacturing process. All unwanted connections are then "blown" during the programming process. Bipolar products are programmed using 20 volt pulses between 50 microseconds and 100 milliseconds in duration. During these pulses 100 to 300 milliAmps (mA) of current exist, blowing unwanted fuses one by one. Fuses are blown one at a time so that the heat generated doesn't damage or weaken the IC. Because of the high currents required, bipolar products have to be programmed one at a time. Since physical fuses are blown, a device cannot be programmed more than once.

No fuses are used in the Cypress CMOS EPLD family. Instead, all devices are based on a EPROM cell to facilitate programming. By using an EPROM cell instead of fuses, programming yields of 100% can be expected since all devices can be functionally tested and erased prior to packaging. Therefore, no programming yield loss can be expected by the user. The EPROM cell used by Cypress serves the same purpose as the fuse used in most bipolar PLD devices. Before programming, the AND gates or product terms are connected via the EPROM cells to both the true and complement inputs.

The EPROM cells are programmed using pulses of high voltage that produce 50 mA of programming current. Eight cells are programmed at a time. Because of the lower current levels used, "gang" programming of 10 to 20 devices in parallel is possible. When the EPROM cell is programmed, that input to an AND gate (or "product term") is disconnected. Programming alters the transistor threshold of the cell so that no conduction can occur, which has the effect of disconnecting the input from product terms. This is equivalent to "blowing" the fuses of a bipolar device, except that exposure to ultraviolet light returns the cell's threshold to normal, effectively erasing the device. Selective programming of EPROM cells enables the specific logic function to be implemented by the user.

Cypress also offers the highest performance silicon PLDs available in ECL technology. Aspen Semiconductor Corporation, a subsidiary of Cypress Semiconductor, has developed a series of bipolar ECL PLDs using an advanced process that incorporates proven Ti-W fuses. Maximum input to output propagation delays of 3 to 6 nanoseconds are achieved with these devices.

PLD Notation and Fusemaps

Logic diagrams have been provided for the various parts in the Cypress Data Book, and the PLD ToolKit Manual. Cypress' logic diagrams employ a common logic convention that is easy to use. *Figure 1* shows the adopted convention. In *Figure 1*, an "X" represents an unprogrammed EPROM cell that is used to connect an input term (corresponding to a vertical line on the logic diagram) to the input of the AND operation (or product term) that is represented by a horizontal line. No "X" means that the EPROM cell on that connection has been programmed or disconnected. The convention adopted does not imply that the input terms are connected on the common line that is indicated, rather that they are being "wire-ANDed." A further extension of this convention is shown in *Figure 2*, which shows the implementation of a simple transfer function. The traditional representation of the same function is shown in *Figure 3*.

Figure 4 is the logic diagram for the PALC16L8. As mentioned earlier, all vertical lines in the array are connected to an array input. These inputs come from the input pins and the I/O pins. Each horizontal line is a "wired-AND" function, also known as a "product term."

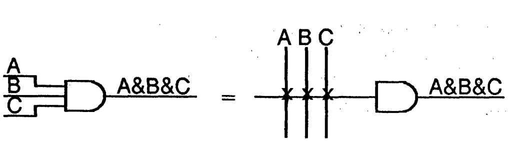


Figure 1. PLD Logic Notation

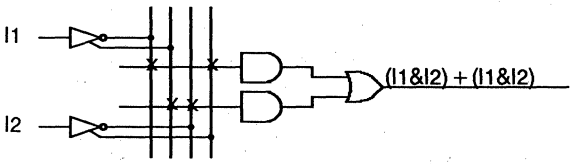


Figure 2. Transfer Function in PLD Logic Notation

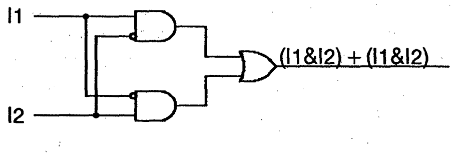


Figure 3. Conventional Schematic of Transfer Function in Figure 2.

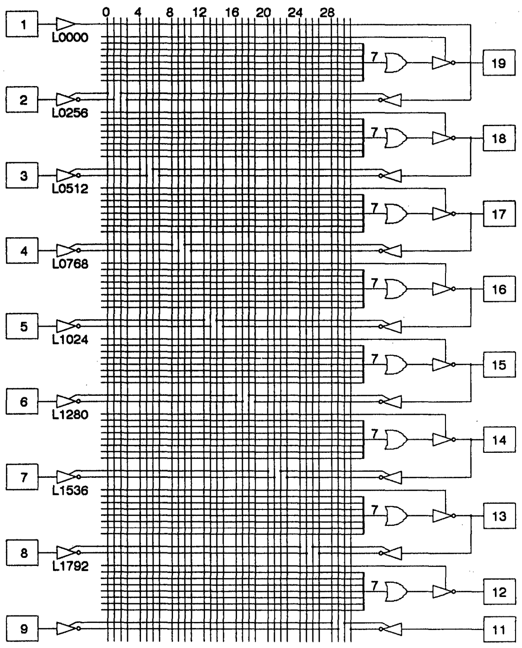


Figure 4. The 16L8 Block Diagram.

The product terms are either connected to the output enable of an output driver, or they are one of seven inputs to an OR gate that connects to the output driver. At each intersection of an input and product term is an EPROM cell. These cells are numbered, starting with 0 as the top left fuse, increasing to the right, and then down. Thus in *Figure 4* cell 0 is the intersection between pin 2, noninverted, and the output enable product term for pin 19. Cell 32 is the intersection between pin 2, noninverting, and the first product term for pin 19. The numbering proceeds until cell 2047, which is the intersection of pin 11, inverted, and the seventh product term for pin 12.

```
Cypress C16L8 Jedec file
File: PLDAPP3.JED produced: 5/2/1989
From Cypress PLD Toolkit CPT100*
*QF2048*  G0*N Security bit Unprogrammed*
L00000 11111111111111111111111111111110*N OE PT, pin= 19*
L00032 10011111111111111111111111111111*N Sum PT, pin= 19*
L00064 01101111111111111111111111111111*N Sum PT, pin= 19*
L00096 00000000000000000000000000000000*N Sum PT, pin= 19*
L00128 00000000000000000000000000000000*N Sum PT, pin= 19*
L00160 00000000000000000000000000000000*N Sum PT, pin= 19*
L00192 00000000000000000000000000000000*N Sum PT, pin= 19*
L00224 00000000000000000000000000000000*N Sum PT, pin= 19*
L00256 00000000000000000000000000000000*N OE PT, pin= 18*
L00288 00000000000000000000000000000000*N Sum PT, pin= 18*
L00320 00000000000000000000000000000000*N Sum PT, pin= 18*
L00352 00000000000000000000000000000000*N Sum PT, pin= 18*
L00384 00000000000000000000000000000000*N Sum PT, pin= 18*
L00416 00000000000000000000000000000000*N Sum PT, pin= 18*
L00448 00000000000000000000000000000000*N Sum PT, pin= 18*
L00480 00000000000000000000000000000000*N Sum PT, pin= 18*
L00512 00000000000000000000000000000000*N OE PT, pin= 17*
L00544 00000000000000000000000000000000*N Sum PT, pin= 17*
L00576 00000000000000000000000000000000*N Sum PT, pin= 17*
L00608 00000000000000000000000000000000*N Sum PT, pin= 17*
L00640 00000000000000000000000000000000*N Sum PT, pin= 17*
L00672 00000000000000000000000000000000*N Sum PT, pin= 17*
L00704 00000000000000000000000000000000*N Sum PT, pin= 17*
L00736 00000000000000000000000000000000*N Sum PT, pin= 17*
L00768 00000000000000000000000000000000*N OE PT, pin= 16*
L00800 00000000000000000000000000000000*N Sum PT, pin= 16*
L00832 00000000000000000000000000000000*N Sum PT, pin= 16*
L00864 00000000000000000000000000000000*N Sum PT, pin= 16*
L00896 00000000000000000000000000000000*N Sum PT, pin= 16*
L00928 00000000000000000000000000000000*N Sum PT, pin= 16*
L00960 00000000000000000000000000000000*N Sum PT, pin= 16*
L00992 00000000000000000000000000000000*N Sum PT, pin= 16*
L01024 00000000000000000000000000000000*N OE PT, pin= 15*
L01056 00000000000000000000000000000000*N Sum PT, pin= 15*
L01088 00000000000000000000000000000000*N Sum PT, pin= 15*
L01120 00000000000000000000000000000000*N Sum PT, pin= 15*
L01152 00000000000000000000000000000000*N Sum PT, pin= 15*
L01184 00000000000000000000000000000000*N Sum PT, pin= 15*
L01216 00000000000000000000000000000000*N Sum PT, pin= 15*
L01248 00000000000000000000000000000000*N Sum PT, pin= 15*
L01280 00000000000000000000000000000000*N OE PT, pin= 14*
L01312 00000000000000000000000000000000*N Sum PT, pin= 14*
L01344 00000000000000000000000000000000*N Sum PT, pin= 14*
L01376 00000000000000000000000000000000*N Sum PT, pin= 14*
L01408 00000000000000000000000000000000*N Sum PT, pin= 14*
L01440 00000000000000000000000000000000*N Sum PT, pin= 14*
L01472 00000000000000000000000000000000*N Sum PT, pin= 14*
L01504 00000000000000000000000000000000*N Sum PT, pin= 14*
L01536 00000000000000000000000000000000*N OE PT, pin= 13*
L01568 00000000000000000000000000000000*N Sum PT, pin= 13*
L01600 00000000000000000000000000000000*N Sum PT, pin= 13*
L01632 00000000000000000000000000000000*N Sum PT, pin= 13*
L01664 00000000000000000000000000000000*N Sum PT, pin= 13*
L01696 00000000000000000000000000000000*N Sum PT, pin= 13*
L01728 00000000000000000000000000000000*N Sum PT, pin= 13*
L01760 00000000000000000000000000000000*N Sum PT, pin= 13*
L01792 00000000000000000000000000000000*N OE PT, pin= 12*
L01824 00000000000000000000000000000000*N Sum PT, pin= 12*
L01856 00000000000000000000000000000000*N Sum PT, pin= 12*
L01888 00000000000000000000000000000000*N Sum PT, pin= 12*
L01920 00000000000000000000000000000000*N Sum PT, pin= 12*
L01952 00000000000000000000000000000000*N Sum PT, pin= 12*
L01984 00000000000000000000000000000000*N Sum PT, pin= 12*
L02016 00000000000000000000000000000000*N Sum PT, pin= 12*
C0B65*
0000
```

Figure 5. A 16L8 JEDEC Map.

A "fuse map" is a software representation of the array of fuses in a programmable logic device. It is an array of binary digits, arranged so that each digit corresponds to a cell in the device. For the PALC16L8 pictured in *Figure 4,* this array is 32 x 64. If a fuse is to be "programmed" or disconnected, the corresponding digit is a 1. If the fuse is to be left intact, the corresponding digit is a 0. A virgin device has all cells conducting, or unprogrammed, so its fuse map is all 0s. A product term, or horizontal line of all zeros, is logically false because it is the AND of the true and complement of each input. If a product term is all 1s, there is no conducting path because all fuses are programmed, and thus nonconductive. This allows the product term to be continuously at an asserted state.

The official, standardized version of a fuse map is called a JEDEC map. This may contain various informational fields and/or comments in addition to the 1s and 0s. The JEDEC map that implements the function in *Figures 2* and *3* is shown in *Figure 5.* The numbers in the leftmost column starting with "L" are the first fuse number in that row. An "N" denotes a note or comment. "QF" precedes the total number of fuses in this device, so it is QF2048 in this example. "F0" means the fuse default is 0 or unprogrammed. "G0" specifies an unprogrammed security fuse, whereas "G1" would denote a programmed security fuse (More on this later). "C" precedes a checksum value for the file. An "*" specifies the end of a field. This file can also contain test vectors, which are not shown here. For more information on the JEDEC Standard, refer to "JEDEC Standard No.3-A, Standard Data Transfer Format Between Data Preparation System and Programmable Logic Device Programmer." This document is available from:

Solid State Products Engineering Council
2001 Eye Street N.W.
Washington D.C. 20006

Most PLD design packages compile the design and translate it into a JEDEC map. This map is then downloaded to the programming hardware, which programs the device(s) accordingly.

First-Generation PLDs

The first PLDs were strictly combinatorial logic with tristate outputs, like the PALC16L8. Then D flip-flops, a clock input, and internal feedback were added, allowing sequential logic, or state machines to be implemented in a single PLD. The 16L8, 16R4 (4 registered outputs), 16R6 (6 registered outputs) and 16R8 (8 registered outputs) became industry standard parts.

Testability was a problem in some of the earlier devices. Since a blank device had all fuses intact, output enables were all turned off, making all pins of the device inputs. This made blank checks difficult. It was also difficult to tell if the fuses could be blown without actually blowing any of them.

To get around these problems, a "Phantom Array" was added to the device. For the 16L8, there are 256 additional bits in the phantom array. These are used to test the PLD functionally and to verify dynamic (AC) operation after the chip is packaged, without using the normal array. The phantom array is usually programmed and verified as part of the final electrical test procedure during the manufacturing process. This verifies both the PLD programmability and functionality. The phantom array may be used by the customer as part of an incoming inspection. The phantom array is so named because the device must be in a special mode to access it. It is not "seen" in regular operating mode. Cypress's EPLDs are also programmed, tested, and then erased before they are packaged.

Another feature that has been added is register preload. The preload function is used to load data into the registers (of registered devices) for testing purposes. This significantly simplifies and shortens the testing procedure. Illegal state resolution can be checked using this feature. Preloading is accomplished by applying a supervoltage (13.5 Volts) pulse of at least 100 microseconds duration to a specific pin, while a second pin is held at VIH. The supervoltage acts as a write strobe and data applied to the I/O pins is clocked into the corresponding registers.

A security fuse was also added as a standard feature. In addition to "writing" a fusemap into a device, any good device programmer is capable of reading a device's fusemap. One of the advantages of a PLD is that the logic in the device is hidden from the observer. This helps keep proprietary portions of a design secret. If the user does not want their PLDs to be read by a programmer, they can program the security bit, which disconnects the lines that are used to verify the array. In a Cypress EPLD, the security EPROM cell has been designed to retain its charge longer than any of the other cells in the array.

The Programmable Macrocell

The basic 20 pin devices still had some limitations. There was no way to control output pin polarity without doing DeMorgan operations on the equations. Quite often the DeMorgan version would have too many product terms to fit in the device, even after several hours of crunching using a logic optimization program.

A variety of the basic 20 pin devices and/or their 24 pin equivalents had to be stocked in order to get the best fit for a given design. Often there were extra registers left unused when the design was finished. Even though the early PLDs tended to be pin limited, the pins associated with those extra registers ended up being wasted because they couldn't be used for anything else.

Enter the 22V10. The 22V10 is a 24 pin device that revolutionized PLDs by introducing the programmable macrocell (See *Figure 6*). The programmable macrocell allows the user to select one of four output configurations: combinatorial, inverting, combinatorial noninverting, registered inverting, and registered noninverting. The pin may be used as an input or bidirectional if the macrocell is specified as combinatorial. Each of the 10 I/O pins has all four options. The option is selected using two fuses, or cells, identical to those in the array. These 20 bits (two for each of 10 macrocells) are added to the bottom of the fusemap that represents the array.

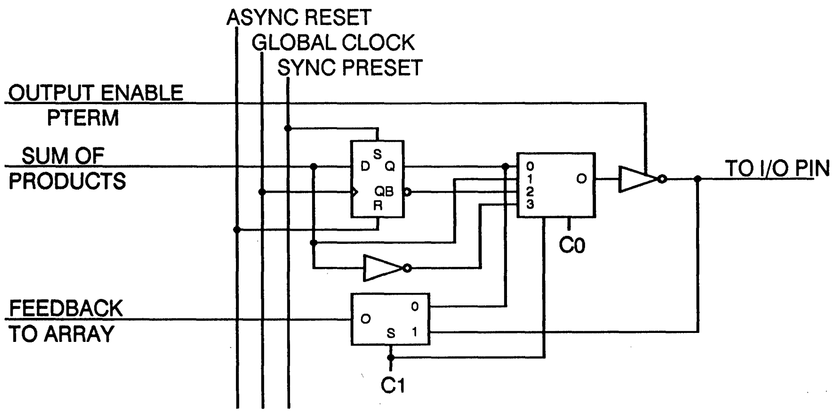


Figure 6. The 22V10 Macrocell.

Another innovation of the 22V10 is that some pins have a larger sum-of-products than others. This is called "Variable Product Term Distribution". In the 22V10, I/O pins have sums from 8 to 16 product terms wide. This accommodates applications such as D flip-flop counters, where several outputs require a large number of product terms.

The 22V10 made yet another improvement. In the 16R8, for example, the device powers up with all registers in reset state. The only way this can be changed is by clocking in new data. The 22V10 added 2 extra product terms: one performs a preset of all registers, the other performs a reset on all registers. Since they are product terms, the preset and reset can be programmed to be the AND of any array input(s). For additional flexibility, the set is designated as a synchronous operation, and the reset is asynchronous.

Because of its flexibility, the 22V10 has become something of an industry standard. It is available in TTL, CMOS, and GaAs. Many companies have introduced similar architectures with slightly different features. For example, the Cypress PLDC20G10 uses a similar macrocell that adds the capability to choose between a product term output enable and a pin controlled output enable. In an effort to make the PLDC20G10 faster and less expensive than the 22V10, the array has been reduced to nine product terms per I/O macrocell, and the preset and reset product terms have been removed.

Another device that was introduced around the same time is the 20RA10, which was targeted for asynchronous registered applications. Like the 22V10, the 20RA10 has I/O pins with programmable polarity bits. The I/O pins of the 20RA10 can be configured as registered or combinatorial, but this is not done with dedicated fuses. Each I/O pin has a sum of four product terms connected, through a polarity switch, to the D input of a flip-flop. Each flip-flop has dedicated product terms connected to its clock, preset, and reset functions. When both the preset and reset of a flip-flop

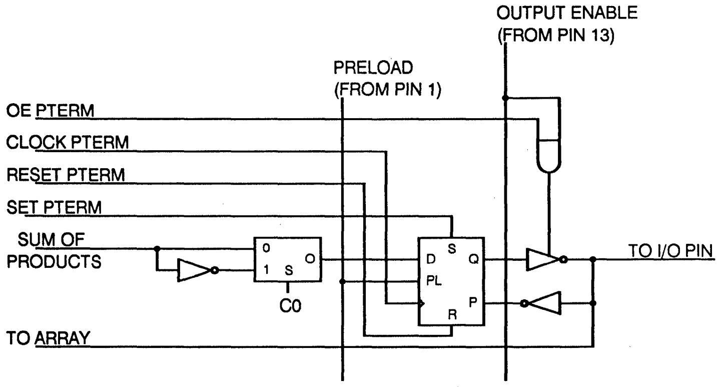


Figure 7. The 20RA10 Macrocell.

are asserted (high), the flip-flop becomes transparent, thus making the output combinatorial.

In addition, the 20RA10 has an unusual output enable scheme. Pin 13 is inverted and ANDed with an output enable product term. If pin 13 is high, all I/O pins are at high impedance. The 20RA10 also offers a synchronous register preload in operating mode. When pin 1 goes low, any data driven onto an I/O pin is latched into the corresponding flip-flop. An I/O pin of the 20RA10 is pictured in *Figure 7*. The flexibility and asynchronous nature of this device make it ideal for bus arbiter, and interrupt controller applications.

Second-Generation PLDs

The architectural features introduced by the 22V10 greatly enhanced PLD flexibility, but there were still some limitations. PLDs still offered only D-type flip-flops, which are cumbersome for some applications, such as counters. Each flip-flop and its feedback still used a pin, even if the flip-flop's output was not needed external to the PLD. Bidirectional, registered pins could not be implemented. High-speed applications often required flip-flops to latch data before the input of the PLD because of the relatively long set-up time, due to propagation delay, for output flip-flops.

Cypress solved all of these problems in the architecture of the CY7C330. In addition to the output registers on the I/O pins, each pin (save power and ground) contains an input register that has a choice of two clocks. This input macrocell makes the 28 pin CY7C330 ideal for pipelined control, and high-speed state machine applications.

Another added feature is the ability to emulate T and JK type flip-flops in the CY7C330. This is very useful in counter designs. In each I/O macrocell, the sum-of-products from the array drives one input of an exclusive OR (XOR) gate. The second input to the XOR gate is another product term. The output of this gate is connected to the D input of the output flip-flop in the macrocell (see *Figure 8*). If the Q output of the flip-flop is fed back and connected to the single product term driving the XOR gate, the sum-of-products acts as the T input of a T type flip-flop. A JK flip-flop can also be emulated in this way, using the relation T=J!Q+KQ. Of course, if a D-type flip-flop is all that is required, the XOR gate can be used to control polarity.

Close examination of *Figure 8* reveals two paths into the array. The first is a multiplexer that selects feedback from either the register or from the Q output of the input register. This is called the "feedback mux." The second path is called the "shared input mux." The inputs to this mux are the Q outputs of input registers belonging to adjacent I/O macrocells. This allows the user to feed back the Q output of a macrocell's output register, and still utilize the pin associated with that macrocell as an input. This, of course can only be done with 6 of the 12 I/O macrocells. If more registers are

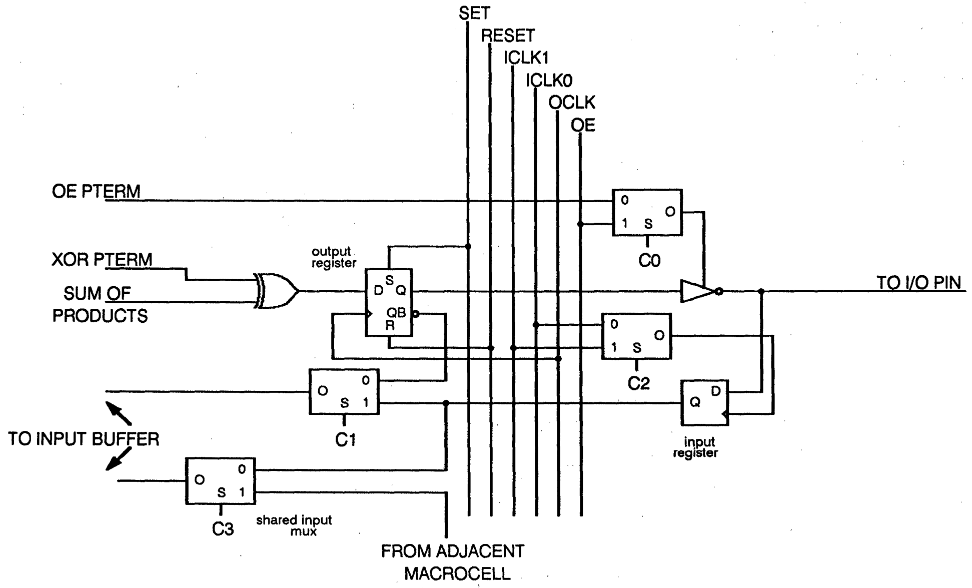


Figure 8. The CY7C330 Macrocell

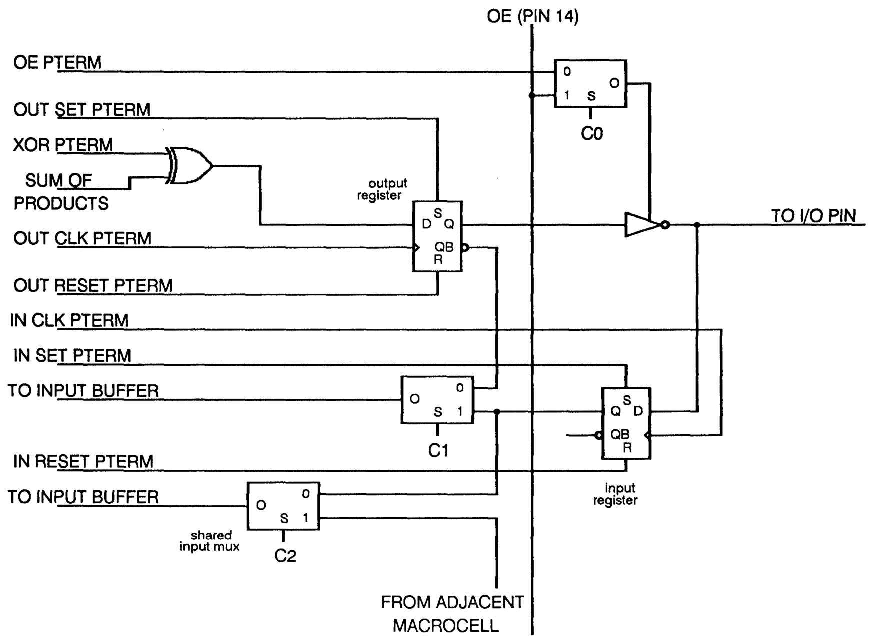


Figure 10. The CY7C331 Macrocell.

needed for an application, there are 4 additional "buried macrocells" in the CY7C330. These are identical to the output register portion of the I/O macrocell, except they are not connected to any pin.

The 20RA10 has many of the same limitations as the 22V10. An additional limitation is that the sum of products is only four wide. Just as the CY7C330 can be considered as an extended, enhanced version of the 22V10, so is the CY7C331 an extension of the 20RA10. The CY7C331 has 12 I/O macrocells. In addition to the 20RA10-like output flip-flops, there are identical flip-flops in the input path. As in the 20RA10, each flip-flop has a product term controlled clock, preset and reset. If the preset and reset product terms are both asserted, the flip-flop becomes transparent. The 20RA10 polarity fuse has been replaced by an XOR gate, which has as inputs the sum of products and a dedicated product term. Thus the polarity of the output can be controlled, or the flip-flops can emulate T or JK flip-flops as in the CY7C330. The CY7C331 macrocell is pictured in *Figure 9.*

Like the 22V10 and CY7C330, the CY7C331 has variable product term distribution with sums from 4 to 12 product terms wide. The CY7C331 has borrowed the shared input mux and output enable schemes from the CY7C330. The operating mode preload in the 20RA10 is not supported in the CY7C331, however the registers can be preloaded using a supervoltage. The CY7C331 has been designed especially for self-timed applications, such as high-speed I/O interfaces. No other PLD has this capability. The CY7C331 is able to support self-timed designs because clock inputs are programmable, internal timing relationships are well controlled, and metastable resolution is ultra-fast.

Another architectural trend is combinatorial PLDs with registered inputs. These are generally used in sophisticated decoding applications, where the address or data is only stable for a short time. In the past an MSI chip with latches or flip-flops was used to capture transient data, and the latched data was fed into a PLD. Now there are PLDs that feature registers or latches on inputs. The CY7C332 features an input macrocell that can be programmed as combinatorial, registered, or latched. There is a choice of two clocks, and the clock polarity is programmable as well. The CY7C332 I/O macrocell (pictured in *Figure 11*) includes the input macrocell, and a combinatorial output path that includes a variable sum of products driving one input of an XOR gate, and a dedicated product term driving the other input. There is also an output enable mux that allows the output enable to be controlled by a product

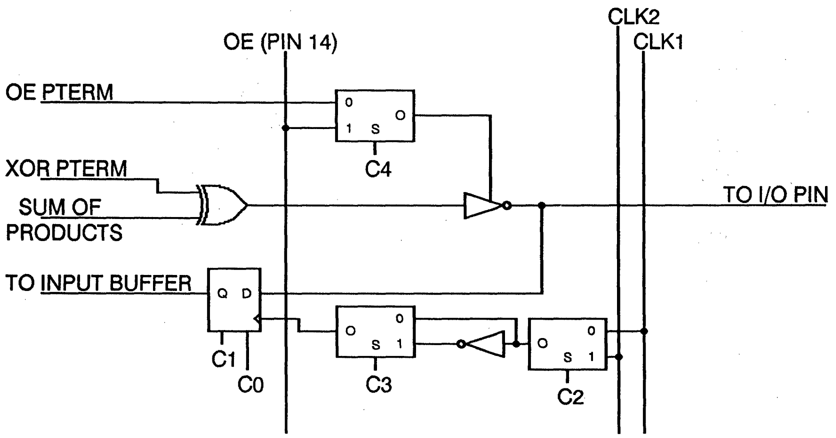


Figure 11. The CY7C332 Macrocell.

term, or pin 14. Of course this combinatorial output path can be used as an input to the programmable input register/latch, thus allowing state machines to be created as well.

High Density PLDs

Because of its low power consumption, higher integration can be achieved with CMOS than with bipolar technologies. Several manufacturers are taking advantage of this, and producing very high-density PLDs. The CY7C342, which is the 68 pin member of the new MAX family, contains 128 flip-flops and over 1000 product terms. Up to 256 additional latches can be configured using Expander Product Terms. The CY7C342 macrocell contains a sum of three product terms driving one input of an XOR. The other XOR input is a dedicated product term. The output of the XOR drives a programmable flip-flop that can be configured as a D, T, JK or SR flip-flop, as well as a latch. There is also a combinatorial path. The flip-flop has asynchronous preset and reset product terms, and a

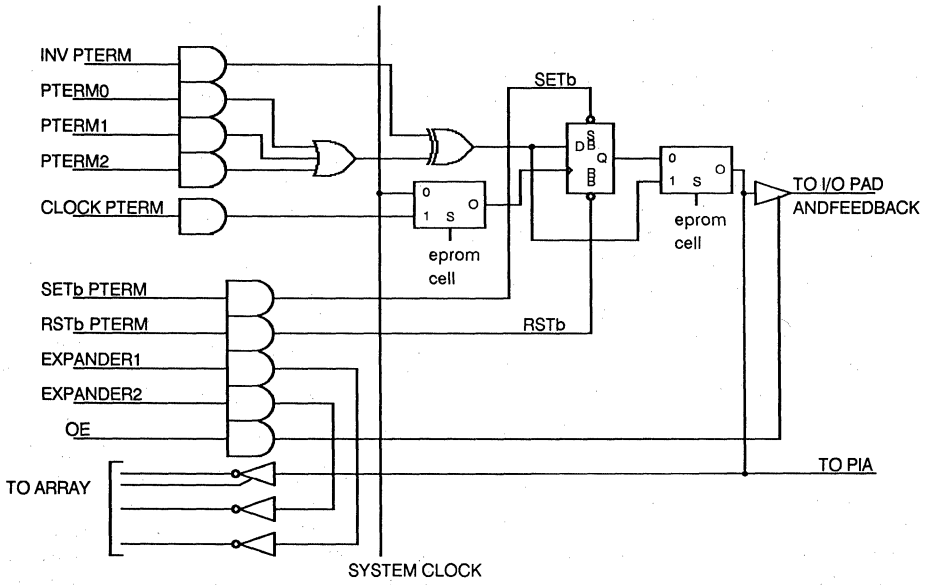


Figure 12. The CY7C342 Macrocell.

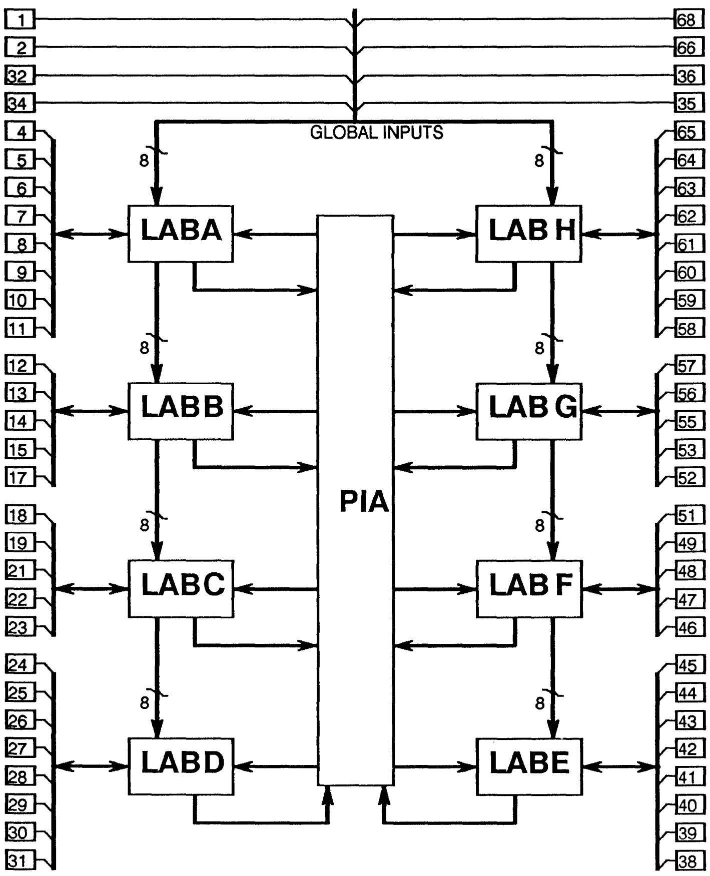


Figure 13. The CY7C342 Block Diagram.

choice of asynchronous clock product term, or a synchronous clock (See *Figure 12*). Macrocells are divided into groups of 16, along with 32 expander product terms. Each of these is called a Logic Array Block or LAB. The CY7C342 contains 8 LABs. These LABs are connected via a Programmable Interconnect Array (or PIA). The block diagram of the CY7C342 is pictured in *Figure 13*. The density, flexibility and speed (typical clock frequency is 50 MHz) allows the CY7C342 to replace over 50 standard TTL devices.

PLD Software Packages

Parts as sophisticated as the MAX chips require equally sophisticated software. The MAX+PLUS™ software offers schematic capture, state machine syntax, Boolean algebra entry, logic reduction, synthesis and fitting, and a timing simulator. Similar packages that support a variety of devices are available from Data I/O and MINC. In addition, OrCad has recently added PLD support to its product line.

More conventional (and less expensive) support is available from ISDATA's LOGiC, Data I/O's ABEL, and Logical Devices' CUPL. These packages offer Boolean equation entry and logic reduction, as well as various higher-level language constructs, state machine syntax, and simulators. All of these packages cover a variety of devices from a variety of vendors.

In addition, most PLD manufacturers offer packages that support only their devices. These packages can be free (PALASM 2) or quite expensive (A+PLUS). Cypress has developed the PLD ToolKit. A basic version that does logic compilation and JEDEC map construction is free. An enhanced package that includes JEDEC read and disassembly capabilities as well as a simulator can be purchased for under $400.00.

CYPRESS SEMICONDUCTOR

Programmable Logic Device Application Brief

Scope and Purpose

The purpose of this application brief is to provide the reader with a basic understanding of Cypress CMOS Programmable Logic Devices. This includes a description of their architecture and design, the technology used in their fabrication, how they are programmed and a discussion of their reliability.

This document will tell the reader how state-of-the-art CMOS technology and a unique architecture have been incorporated in a family of PLD integrated circuits that are functionally equivalent, pin compatible, and superior in performance to their bipolar counterparts.

The appendix discusses and illustrates the design techniques that Cypress uses on all products to eliminate latchup and improve ESD (Electro-Static-Discharge) protection.

Introduction

The PLD is a Programmable Logic Device. The basic (functional) logic structure of a PLD is programmable AND array whose outputs feed into a fixed OR array. The pertinent parameters are the number of inputs, the number of outputs, the width (number of factors) in the AND array and the width (number of terms) in the OR array. The Boolean equation implemented is the sum-of-products or minterm form.

The first PLDs were strictly combinatorial logic. They were followed by devices that added latches (D flip-flops) a clock input, and internal feedback. For the first time a programmable, sequential, state machine could be implemented in a single package. Three-state outputs, the "security fuse", flip-flop initialization, and in general terms "testability" are features that have been added for increased flexibility.

Applications

PLDs are used to replace SSI/MSI logic and "glue chips" primarily to increase packaging density. A single PLD is the functional equivalent of many SSI ICs (in the 200–500 equivalent gate range). When PLDs are used to replace standard logic gates, the resulting reduction in PC card area, although application dependent, has been found to vary between 4 to 1 and 10 to 1. i.e., One PLD will replace between four and ten 14 pin ICs. Secondary benefits to the user are reduced parts inventory, reduced power, higher reliability, faster design and turnaround time, product secrecy and equal (matched) propagation delays through the AND OR array.

Reliability

Reliability studies have shown that system reliability is inversely proportional to the number of interconnections between system elements. However, the failure rate for mature ICs is about 0.1% per thousand hours and has remained constant during the last twenty years in spite of the fact that circuit complexity (density) has increased by more than two orders of magnitude.

The conclusion is that higher levels of IC integration provide increased system reliability. Thus the user is increasing system reliability when Cypress CMOS PLDs replace glue chips.

Programming

PLDs must be programmed. This can be accomplished by either designing and building a programmer or purchasing one for $1,000 to $10,000.

Programming Bipolar PLDs

Bipolar PLDs use a fuse as the programmable element. In an unprogrammed device all of the connections are "made" during the manufacturing process and the unwanted connections are later "unmade" by blowing fuses during the programming process.

Bipolar products are programmed using 20 Volt pulses of durations from 50 microseconds to 10 milliseconds during which 100 to 300 milliamperes (mA) of current exist. In order to limit the heat generated during programming, the duty cycle for the programming pulses is limited to 20 to 30 percent. One fuse is blown at a time so that the heat generated will neither permanently damage the IC nor stress it to the point that it could fail later. Some programming algorithms take into account the physical locations of the fuses and avoid sequentially blowing fuses that are physically close to each other in order to prevent excessive localized heating of the chip. Because of the high currents required, bipolar products are not "gang" programmed, as are EPROMs.

Programming Cypress CMOS PLDs

Cypress PLDs are programmed by storing charge on the floating gate of an EPROM transistor. Charge storage is accomplished by hot carrier injection; a process that does not physically destroy material or heat the device. During programming, EPROM cells are stressed significantly less than fuses. In addition, every cell is programmed, tested and erased as part of the manufacturing process. This 100% testing guarantees a very high programming yield to the customer, which is impossible to guarantee with any fuse programmable device.

The storage mechanism is well understood. Products using it have been in volume production for more than ten years. Reliability studies have been performed by many independent organizations and all have concluded that the technology is reliable.

Cypress PLDs are programmed using high voltage pulses of durations from 100 microseconds to 10 ms, during which 50 milliamperes of programming current exist. Eight bits are programmed at the same time and, because of the lower currents required, gang programmers that can handle 10 to 20 devices in parallel are possible.

Before programming, AND gates or PRODUCT TERMS are connected via EPROM cells to both true and complement inputs. Programming an EPROM cell disconnects an input from a PRODUCT TERM. Selective programming of these cells enables a specific logic function to be implemented. PLDs are supplied in a number of functional configurations. These functional variations offer the user the choice of combinatorial as well as registered paths to implement logic functions.

CMOS Technology

Cypress PLDs are fabricated using an advanced "N-well" CMOS technology. The use of proven EPROM technology to achieve memory non-volatility, combined with novel circuit design and a unique architecture, provides the user with a superior product in terms of performance, reliability, testability and programmability.

PAL® is a registered trademark of Monolithic Memories, Inc.

Functional Description

General

The variations of PLD functions available are listed in Table 1. The 16L8, which is used as an example (*see Figure 2*), is purely combinatorial and consists of eight groups of 7-input AND gates, each of which can have up to 32 inputs. One of the AND gates of each group (of 8) is used to enable the (inverting) output driver, so that 7 AND gates (each of which may have 32 inputs) each feed one OR gate, whose output is inverted.

The 16R8 is similar to the 16L8, except that the outputs are latched using D flip-flops (with a common clock), the inputs to the 8 OR gates are the outputs of 8 AND gates; the three-state output drivers are enabled by a common enable input.

The reader should refer to the PLD data sheets for a more detailed description of the other members of the family. The 16R4, 16R6 and 16R8 have 4, 6, or 8 registered outputs with feedback.

The 22V10 offers a unique macro-cell flexibility to allow any combination of up to 10 combinatorial or registered outputs. In a similar manner the 20G10 uses macro-cells to allow the user to program the functionality of the 10 most popular PAL® 24 devices.

Register Preload

The preload function is used to load data into the internal register (of registered devices) for testing purposes. This significantly simplifies and shortens the testing procedure. Loading is accomplished by applying a supervoltage pulse of at least 100 microseconds duration to pin 5 as a write pulse while pin 11 is held at VIH and data is applied to pins 12 through 19.

Security Function

The security function prevents the contents of the regular array from being electrically verified. This enables the user to safeguard proprietary logic. The EPROM technology prevents the state of the cell from being visually ascertained. The security function is implemented by programming an EPROM cell that disconnects the lines that are used to verify the array. This cell has been designed to retain its charge longer than any of the other cells in the array.

Commercial Selection Guide

Generic Part Number	Logic	Output Enable	Outputs	I_{CC} mA		t_{PD} ns		t_S ns		t_{CO} ns	
				L	STD	-25	-35	-25	-35	-25	-35
16L8	(8) 7-wide AND-OR-Invert	Programmable	(6) Bidirectional (2) Dedicated	45	70	25	35	—	—	—	—
16R8	(8) 8-wide AND-OR	Dedicated	Registered Inverting	45	70	—	—	20	30	15	25
16R6	(6) 8-wide AND-OR	Dedicated	Registered Inverting	45	70	25	35	20	30	15	25
	(2) 7-wide AND-OR-Invert	Programmable	Bidirectional								
16R4	(4) 8-wide AND-OR	Dedicated	Registered Inverting	45	70	25	35	20	30	15	25
	(4) 7-wide AND-OR-Invert	Programmable	Bidirectional								
20G10	(10) 8-wide AND-OR-Invert with MACRO	Programmable or Dedicated	Programmable Bidirectional or Registered	—	55	25	35	15	30	15	25
22V10	(10) variable AND-OR-Invert with MACRO	Programmable	Programmable Bidirectional or Registered	55	90	25	35	15	30	15	25

Military Selection Guide

Generic Part Number	Logic	Output Enable	Outputs	V_{CC} mA	t_{PD} ns				t_S ns				t_{CO} ns			
					-20	-25	-30	-40	-20	-25	-30	-40	-20	-25	-30	-40
16L8	(8) 7-wide AND-OR-Invert	Programmable	(6) Bidirectional (2) Dedicated	70	20	NA	30	40	—	NA	—	—	—	NA	—	—
16R8	(8) 8-wide AND-OR	Dedicated	Registered Inverting	70	—	NA	—	—	20	NA	25	35	15	NA	20	25
16R6	(6) 8-wide AND-OR	Dedicated	Registered Inverting	70	20	NA	30	40	20	NA	25	35	15	NA	20	25
	(2) 7-wide AND-OR-Invert	Programmable	Bidirectional													
16R4	(4) 8-wide AND-OR	Dedicated	Registered Inverting	70	20	NA	30	40	20	NA	25	35	15	NA	20	25
	(4) 7-wide AND-OR-Invert	Programmable	Bidirectional													
20G10	(10) 8-wide AND-OR-Invert with MACRO	Programmable	Programmable Bidirectional or Registered	80	NA	—	30	40	NA	—	25	35	NA	—	20	25
22V10	(10) variable AND-OR-Invert with MACRO	Programmable	Programmable Bidirectional or Registered	100	NA	25	30	40	NA	20	25	35	NA	20	20	25

Table 1. PLD Selection Guide

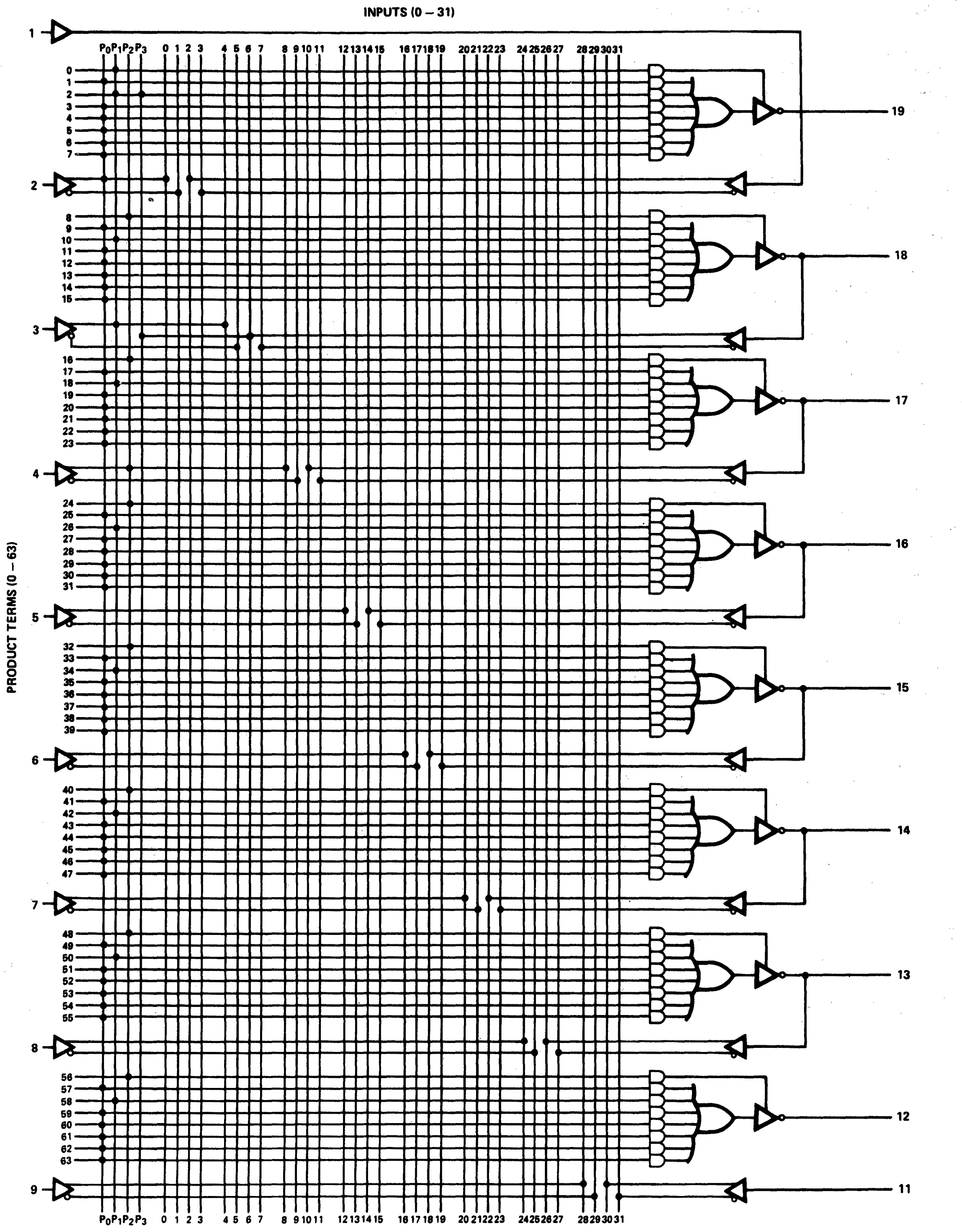

INPUTS (0 – 31)
PRODUCT TERMS (0 – 63)
P0P1P2P3 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
1
2
3
4
5
6
7
8
9
19
18
17
16
15
14
13
12
11
0049–2

Figure 2. Functional Logic Diagram PAL C 16L8A

There are 2048 EPROM cells in the PAL C 20 array that are used to specify up to 32 inputs for 8 groups of 7-input AND OR gates and 8 32-input AND output enable gates. In normal usage, a maximum of 16 inputs would be connected to any AND gate, because connecting both a true and a complement input of the same signal to the input of an AND gate will result in a constant LOW output.

Phantom Array

There are an additional 256 bits in a phantom array that are used to test the PAL C 20 device functionally and to verify dynamic (AC) operation without using the regular array after the device is packaged. The phantom array is programmed and verified as part of the final electrical test procedure during the manufacturing process. It may be used by the customer as part of an incoming inspection and could be used to verify programmability as well as functionality. Three input pins are used to verify operation of the phantom array. One (pin 2) has a worst case (longest physical length) propagation delay path through the regular array.

Programming the Arrays

The phantom array is programmed in the same manner as the regular array. Both are addressed as byte arrays for programming. The normal array has 256 bytes to program and the phantom array has 32 bytes. The customer may test the programmed phantom array functionally and dynamically as part of an incoming inspection.

Programming the EPROM Cell

A schematic of the two-transistor EPROM cell used in all PLDs is illustrated in *Figure 1.* Conventional EPROMS use one transistor per cell and its design is a compromise between being able to program (write) rapidly and read. Cypress uses a two-transistor cell that enables the PLDs to achieve superior performance by optimizing the read transistor, R, and program transistor, P, for their respective functions. The cell size is 20.4 microns by 6.7 microns. Note that the selection gates, the floating gates and the sources of both transistors are (respectively) connected together.

Operation

In the unprogrammed state, the threshold voltage of the R transistor is less than that of the P transistor.

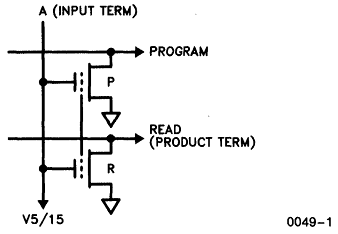


Figure 1. PLD EPROM Cell Schematic

To program the cell, the input line (A) is raised to 15 volts, which causes charge to be stored on the floating gate of the P transistor, which causes its threshold to increase by approximately 7 volts. Because the floating gates of both transistors are connected together, the threshold of the R transistor increases by the same amount.

To read from the cell, the input line (A) is raised to 5 volts. If the cell had been programmed, this voltage would not be sufficient to turn-on the read transistor. However, if the cell had not been programmed, the read transistor would turn-on. Under this condition the current through the read transistor is 150 microamperes; approximately an order of magnitude greater than that used in a conventional EPROM cell. The larger current is required in order to achieve the specified performance.

Operational Overview

The device operates in two basic modes; normal and PROGRAM. In the normal mode either the Regular array or the Phamtom array may be used, together with the data inputs, to determine the state of the outputs. In the PROGRAM mode either the Regular array or the Phantom array may be programmed using the 8 outputs (pins 12–19) as data inputs and pins 2 through 9 as address inputs.

Table 2 illustrates the various modes of operation for the PAL C 20 device. They are decoded by high-voltage-sensitive on-chip circuits. It is permitted to go from any mode to any other mode. Note that the normal data output pins (12–19) are used as data input pins for programming.

Programming

Tables 3 and 4 indicate how the regular and the phantom arrays in the PAL C 20 device are addressed. The 20G10 and 22V10 are similar. The regular array is addressed as a

256 word (8 × 32) by 8-bits per word memory. The phantom array is selected using the same addresses as columns 0, 1, 2 and 3, but with pin 7 at Vpp (per Tables 2 and 4).

In either case (normal or phantom array), the product terms are addressed in groups of 8 as shown in Table 3. There is a one-to-one correspondence between the data to be programmed and the D0–D7 inputs and the product terms, as modified modulo 8, by the address on pins 2, 3, 4 (Refer to *Figure 2*). In other words, a one on D0 corresponds to de-selecting the "product term input" at input line 0 and product term 0. A one on D1 corresponds to de-selecting the product term input at input line 0 and product term 8, etc. One method of programming the array would be to program and verify the bits corresponding to the first product term address and then increment a counter that generates the "OR" gate addresses (pins 2, 3, 4) and then program and verify the second row of Table 3, and continue this process 8 times until all 64 product terms associated with input line 0 have been programmed and verified. To select the second (1) input term, address pins 6, 7, 8 and 9 are held LOW (as before) and pin 5 = HIGH. The preceding sequence is then repeated 31 more times, incrementing pins 5 through 9 in a binary sequence, to program and verify the entire array. The other members of the family are programmed in an identical manner.

Table 2. PAL C 20 Series Operating Modes

Pin Name	Vpp	PGM/$\overline{\text{OE}}$	A1	A2	A3	A4	A5	D7–D0	Notes
Pin Number	(1)	(11)	(3)	(4)	(5)	(6)	(7)	(12–19)	
Operating Modes									
PAL	X	X	X	X	X	X	X	Programmed Function	3, 4
Program PAL	V_{PP}	V_{PP}	X	X	X	X	X	Data In	3, 5
Program Inhibit	V_{PP}	V_{IHP}	X	X	X	X	X	High Z	3, 5
Program Verify	V_{PP}	V_{ILP}	X	X	X	X	X	Data Out	3, 5
Phantom PAL	X	X	X	X	X	V_{PP}	X	Programmed Function	3, 6
Program Phantom PAL	V_{PP}	V_{PP}	X	X	X	X	V_{PP}	Data In	3, 7
Phantom Program Inhibit	V_{PP}	V_{IHP}	X	X	X	X	V_{PP}	High Z	3, 7
Phantom Program Verify	V_{PP}	V_{ILP}	X	X	X	X	V_{PP}	Data Out	3, 7
Program Security Bit	V_{PP}	V_{PP}	V_{PP}	X	X	X	X	High Z	3
Verify Security Bit	X	X	(Note 8)	V_{PP}	X	X	X	High Z	3
Register Preload	X	X	X	X	V_{PP}	X	X	Data In	3, 9

Notes:
1. V_{PP} = 13.5 ±0.5V, I_{PP} = 50 mA; V_{CCP} = 5 ±0.25V; V_{IHP} = 3V; V_{ILP} = 0.4V.
2. Measured at 10% and 90% points.
3. $V_{SS} < X < V_{CCP}$.
4. All "X" inputs operational per normal PAL function.
5. Address inputs occupy Pins 2 thru 9 inclusive, for both programming and verification see programming address *Tables 3* and *4*.
6. All "X" inputs operational per normal PAL function except that they operate on the function that occupies the phantom array.
7. Address inputs occupy Pins 2 thru 9 inclusive, for both programming and verification see programming address *Tables 3* and *4*. Pin 7 is used to select the phantom mode of operation and must be taken to V_{PP} before selecting phantom program operation with V_{PP} on Pin 1.
8. The state of Pin 3 indicates if the security function has been invoked or not. If Pin 3 = V_{OL} security is in effect, if Pin 3 = V_{OH}, the data is unsecured and may be directly accessed.
9. For testing purposes, the output latch on the 16R8, 16R6 and 16R4 may be preloaded with data from the appropriate associated output line.

Table 3. PAL C 20 Series Product Term Addresses

Product Term Addresses										
Binary Address			Line Number							
Pin Numbers										
(4)	(3)	(2)								
V_{ILP}	V_{ILP}	V_{ILP}	0	8	16	24	32	40	48	56
V_{ILP}	V_{ILP}	V_{IHP}	1	9	17	25	33	41	49	57
V_{ILP}	V_{IHP}	V_{ILP}	2	10	18	26	34	42	50	58
V_{ILP}	V_{IHP}	V_{IHP}	3	11	19	27	35	43	51	59
V_{IHP}	V_{ILP}	V_{ILP}	4	12	20	28	36	44	52	60
V_{IHP}	V_{ILP}	V_{IHP}	5	13	21	29	37	45	53	61
V_{IHP}	V_{IHP}	V_{ILP}	6	14	22	30	38	46	54	62
V_{IHP}	V_{IHP}	V_{IHP}	7	15	23	31	39	47	55	63
			D0	D1	D2	D3	D4	D5	D6	D7
			Programmed Data Input							

Table 4. PAL C 20 Series Input Term Addresses

Input Term Addresses					
Input Terms Numbers	Binary Addresses				
	Pin Numbers				
	(9)	(8)	(7)	(6)	(5)
0	V_{ILP}	V_{ILP}	V_{ILP}	V_{ILP}	V_{ILP}
1	V_{ILP}	V_{ILP}	V_{ILP}	V_{ILP}	V_{IHP}
2	V_{ILP}	V_{ILP}	V_{ILP}	V_{IHP}	V_{ILP}
3	V_{ILP}	V_{ILP}	V_{ILP}	V_{IHP}	V_{IHP}
4	V_{ILP}	V_{ILP}	V_{IHP}	V_{ILP}	V_{ILP}
5	V_{ILP}	V_{ILP}	V_{IHP}	V_{ILP}	V_{IHP}
6	V_{ILP}	V_{ILP}	V_{IHP}	V_{IHP}	V_{ILP}
7	V_{ILP}	V_{ILP}	V_{IHP}	V_{IHP}	V_{IHP}
8	V_{ILP}	V_{IHP}	V_{ILP}	V_{ILP}	V_{ILP}
9	V_{ILP}	V_{IHP}	V_{ILP}	V_{ILP}	V_{IHP}
10	V_{ILP}	V_{IHP}	V_{ILP}	V_{IHP}	V_{ILP}
11	V_{ILP}	V_{IHP}	V_{ILP}	V_{IHP}	V_{IHP}
12	V_{ILP}	V_{IHP}	V_{IHP}	V_{ILP}	V_{ILP}
13	V_{ILP}	V_{IHP}	V_{IHP}	V_{ILP}	V_{IHP}
14	V_{ILP}	V_{IHP}	V_{IHP}	V_{IHP}	V_{ILP}
15	V_{ILP}	V_{IHP}	V_{IHP}	V_{IHP}	V_{IHP}
16	V_{IHP}	V_{ILP}	V_{ILP}	V_{ILP}	V_{ILP}
17	V_{IHP}	V_{ILP}	V_{ILP}	V_{ILP}	V_{IHP}
18	V_{IHP}	V_{ILP}	V_{ILP}	V_{IHP}	V_{ILP}
19	V_{IHP}	V_{ILP}	V_{ILP}	V_{IHP}	V_{IHP}
20	V_{IHP}	V_{ILP}	V_{IHP}	V_{ILP}	V_{ILP}
21	V_{IHP}	V_{ILP}	V_{IHP}	V_{ILP}	V_{IHP}
22	V_{IHP}	V_{ILP}	V_{IHP}	V_{IHP}	V_{ILP}
23	V_{IHP}	V_{ILP}	V_{IHP}	V_{IHP}	V_{IHP}
24	V_{IHP}	V_{IHP}	V_{ILP}	V_{ILP}	V_{ILP}
25	V_{IHP}	V_{IHP}	V_{ILP}	V_{ILP}	V_{IHP}
26	V_{IHP}	V_{IHP}	V_{ILP}	V_{IHP}	V_{ILP}
27	V_{IHP}	V_{IHP}	V_{ILP}	V_{IHP}	V_{IHP}
28	V_{IHP}	V_{IHP}	V_{IHP}	V_{ILP}	V_{ILP}
29	V_{IHP}	V_{IHP}	V_{IHP}	V_{ILP}	V_{IHP}
30	V_{IHP}	V_{IHP}	V_{IHP}	V_{IHP}	V_{ILP}
31	V_{IHP}	V_{IHP}	V_{IHP}	V_{IHP}	V_{IHP}
P0	V_{ILP}	V_{ILP}	V_{PP}	X	X
P1	V_{ILP}	V_{IHP}	V_{PP}	X	X
P2	V_{IHP}	V_{ILP}	V_{PP}	X	X
P3	V_{IHP}	V_{IHP}	V_{PP}	X	X

Implementation

A simplified block diagram of a 16L8 is presented in *Figure 3*. The method of programming and sensing is illustrated in *Figure 4*.

Programming Operation

Pins 5–9 are decoded (according to Table 4) in a one of 32 decoder, whose outputs correspond to the inputs labeled 0–31 of *Figure 2*. For programming, 15 volts is applied to the bottom of the input term line through a weak depletion mode device (*Figure 4*). The EN (enable) signal to all of the three-state drivers is LOW, which prevents the normal input signals from driving the input term lines during programming. The D0–D7 inputs (pins 19 through 12) drive the program transistors (0, 8, 16, 24 etc.) as selected by pins 2, 3, 4 and as listed in Table 3. To disconnect an input term line from a product term line, the P transistor is forward biased, which increases the threshold of the R transistor.

Verify Operation

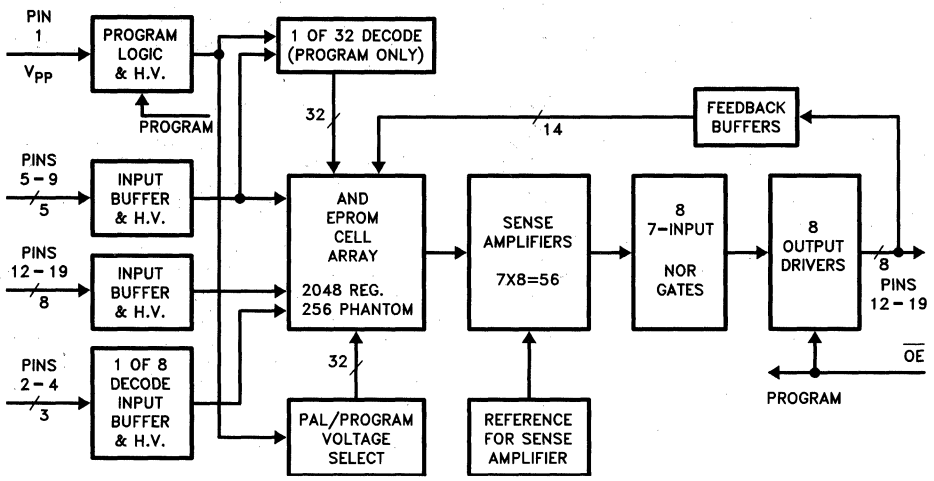
To verify the programmed cells, the device must go from the PROGRAM mode to the PROGRAM INHIBIT mode to the PROGRAM VERIFY mode. This is accomplished by reducing the voltage on pin 11 to VIHP (3V) and then to VILP (0.4V). Internal to the device (see *Figure 4*) the 1 of 32 decoder is disabled, the EN signal is LOW, and 31 of the 32 input term lines are at zero volts. The line being verified is at 5 volts. The input address lines (pins 2 through 9) do not need to change when going from program to verify.

The "ones" that were programmed cause the thresholds of the R transistors to increase, so they do not turn on during verify. Conversely, the unprogrammed transistors do turn on, so the complement (inverse) of the data programmed is read during verify.

Normal Operation

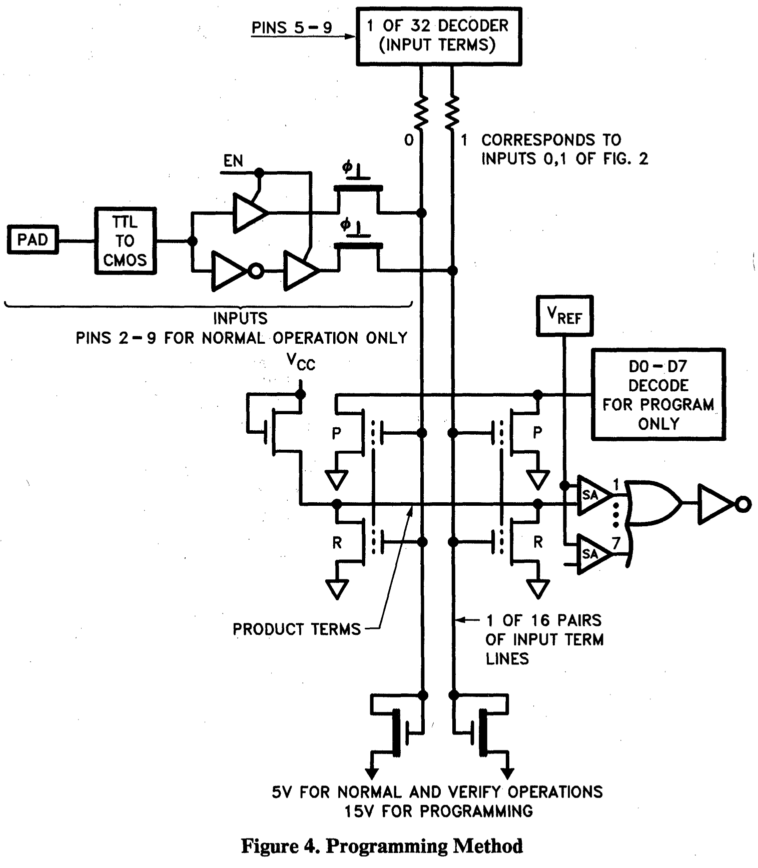
The PAL device will implement the programmed function when there are no supervoltages applied to any of the pins. During regular PAL operation the 1 of 32 decoder and the D0–D7 decoder are disabled, the EN signal is HIGH and all 32 input term lines are at 5 volts. Under these conditions, the data at the input pins is applied to all 64 of the product term lines. If any of the P transistors (16 per product term line) had not been programmed, they will turn on and pull the lower input to the corresponding sense amplifier (SA) to 2 volts or less. This voltage will be less than the reference (Vref) so that the output of the sense amplifier will be LOW.

The reference is an unprogrammed EPROM cell that tracks the same process, voltage and temperature variations that affect all of the cells in the array. It is approximately three volts at room temperature and nominal (5 volts) V_{CC}.



0049-3

Figure 3. 16L8 Device Simplified Block Diagram


PINS 5-9
1 OF 32 DECODER (INPUT TERMS)
0
1
CORRESPONDS TO INPUTS 0,1 OF FIG. 2
EN
ϕ
ϕ
PAD
TTL TO CMOS
INPUTS
PINS 2-9 FOR NORMAL OPERATION ONLY
VCC
VREF
D0-D7 DECODE FOR PROGRAM ONLY
P
P
SA
1
SA
7
R
R
PRODUCT TERMS
1 OF 16 PAIRS OF INPUT TERM LINES
5V FOR NORMAL AND VERIFY OPERATIONS
15V FOR PROGRAMMING

0049-4

Figure 4. Programming Method

Phantom Operation

The PAL C 20 device is in the PHANTOM mode of operation when a supervoltage (V_{PP} = 13.5V) is applied to pin 6. The phantom array is programmed as shown in *Figure 2*. The user may measure the worst case propagation delay from the pin 2 input to the outputs (pins 12 through 17). The truth table for the phantom array is shown in Table 5.

Table 5. Phantom Array Truth Table

	Inputs			Outputs							
Pin	**2**	**3**	**4**	**19**	**18**	**17**	**16**	**15**	**14**	**13**	**12**
	0	0	1	X	X	1	1	1	1	1	1
	1	0	1	X	X	0	0	0	0	0	0
	0	1	X	1	0	X	X	X	X	X	X
	0	1	X	0	1	X	X	X	X	X	X

Reliability

Reliability is designed into all Cypress products from the beginning by using design techniques to eliminate latchup, improve ESD and by paying careful attention to layout. In addition, all products are tested for all known types of CMOS failure mechanisms.

Failure mechanisms can be either classified as those generic to CMOS technology or those specific to EPROM devices.

Table 6 lists both categories of failures, their relevant activation energies, Ea in eV (electron volts), and the detection method used by Cypress. In both cases, the mechanisms are aggravated by HTOL (High Temperature Operating Life) tests and HTS (High Temperature Storage) tests.

Specific EPROM failure mechanisms include charge loss, charge gain and electron trapping. Charge loss is accelerated by thermal energy and field emission effects.

Thermal charge loss failures usually occur on random bits and are often related to latent manufacturing defects.

Field emission effects are generally detected as weakly programmed cells. The high voltages used to program a "selected bit" may disturb (as a result of a defect) an "unselected bit"

Charge gain is due to electrons accumulating on a floating gate as a result of bias or voltage on the gate. This results in a reduced read margin. This mechanism is generally negligible.

Charge gain and charge loss are monitored on every Cypress die in wafer form by programming, performing a HTS test and verifying that the programmed data is retained in the device.

Table 6. Generic CMOS Failure Mechanisms

Mechanism	Activation Energy (eV)	Detection Method
Surface Charge	0.5 to 1.0	HTOL, Fabrication Monitors
Contamination	1.0 to 1.4	HTOL, Fabrication Monitors
Electromigration	1.0	HTOL
Micro-cracks	—	Temperature Cycling
Silicon Defects	0.3	HTOL
Oxide Breakdown	0.3	High Voltage Stress, HTOL
Hot Electron Injection	—	LTOL (Low Temp. Operating Life)
Fabrication Defects	—	Burn In
Latchup	—	High Voltage Stress, Burn In, Characterization
ESD	—	Characterization
Charge Loss	0.8 to 1.4	HTS (High Temperature Storage)
Charge Gain (Oxide Hopping)	0.3 to 0.6	HTOL
Electron Trapping in Gate Oxide	—	Program/Erase Cycle

Notes:
Table 6 has been adapted from, "An Evaluation of 2708, 2716, 2532, and 2732 Types of U-V EPROMS, Including Reliability and Long Term Stability." Danish Research Center for Applied Electronics, Nov. 1980.

HTOL Testing

High Temperature Operating Life test (or burn-in) is used to detect most generic CMOS failure mechanisms. Units are placed in sockets under bias conditions with power applied and at elevated temperatures for a specific number of hours. This test is used to weed out the "weak sisters" that would fail during the first 100 to 500 hours of operation under normal operating temperatures. HTOL tests are also used to measure parameter shifts in order to predict (and screen for) failures that would occur much later.

HTS Testing

High Temperature Storage tests are used to thermally accelerate charge loss. These tests are performed at the wafer level and under unbiased conditions. Both pass/fail data as well as shifts in thresholds may be measured. For a more detailed discussion of charge loss screening the reader is referred to the article on EPROM reliability beginning on page 132 of the August 14, 1980, issue of Electronics magazine.

The generally accepted screening method for identifying charge loss is a 168 hour bake at 250°C. This correlates with more than 220,000 years of normal operation at 70°C using a failure activation energy of 1.4 eV.

Initial Qualification

The process in general and the EPROM cell design in particular was qualified using HTS (bake) at 250°C for 256 hours, in conjunction with an HTOL test at 125°C for 1000 hours.

Procedure

Four wafers were erased using ultraviolet light and the linear thresholds of the cell read transistors measured at twenty-five "sites" on each wafer.

The wafers were then programmed and the linear thresholds then measured and recorded.

The wafers were alternately baked at 250°C and the linear thresholds measured and recorded at 0.25, 0.5, 1, 2, 4, 8, 16, 32, 64, 128, and 256 hours. The number of device hours is then 100 × 256 = 25,600.

Results

The average threshold reduction due to charge loss was 0.66 volts. The range was eight to ten percent of the average initial threshold of 7.7 volts. This reduced threshold is greater than four volts above the sense amplifier voltage reference. There were no failures.

If the charge loss failure activation energy is assumed to be 1.4 eV, the HTS time of 256 hours at 250°C translates to 438,356 years of operation at 70°C.

The time translations were computed using the industry standard Arrhenius equation, which converts the time to failure (operating lifetime) at one temperature and time to another temperature and time.

Summary

Sample size:	100
Device hours:	25,600 hours
HTS conditions:	256 hours at 250°C
Average initial threshold:	7.7 volts
Average threshold decrease:	0.66 volts
Standard deviation:	0.12
Lifetime (1.4 ev):	438,356 years at 70°C

Conclusion

An HTS experiment, performed according to industry standard conditions, and using representative Cypress product confirms that the data retention characteristics of the EPROM cell used in all Cypress PLDs and PROMs guarantees a minimum operating lifetime of 438,356 years for activation energies of 1.4 eV.

Production Screen

Units from the same population were assembled without being subjected to HTS and were subjected to an HTOL of 150 degrees C for 1000 hours. The units were tested at 12, 24, 48, 96, 168, 336, and 1008 hours and the measurements recorded. Variations in the thresholds of the EPROM cells were measured and correlated to the units tested in the HTS test in order to determine a maximum acceptable rate of charge loss in order to guarantee data retention over their normal operating lifetime.

Advantages Over Bipolar

Lower power results in several benefits to the user. They are:

- Lower capacity and, therefore, lower cost power supplies.
- Reduced cooling requirements.
- Increased long term reliability due to lower die junction temperatures.

Power dissipation may be further reduced by driving the inputs between 0.5 volts (or less) and 4 volts (or more). This reduces the power dissipation in the input TTL to CMOS buffers, which dissipate power when their inputs are between 0.8 volt and 3 volts. Each buffer draws approximately 0.8 mA of I_{CC} current at V_{IN} = 2 volts.

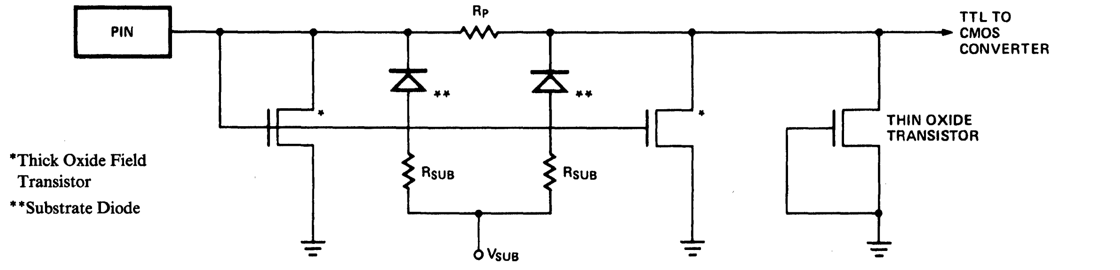


Figure 5. Input Protection Circuit

Appendix

The Cypress double-layer polysilicon, single-layer metal, N-well, CMOS technology has been optimized for performance. Careful attention to design details and layout techniques has resulted in superior performance products with improved ESD input protection and improved latchup protection.

Input ESD Protection

The circuit shown in *Figure 5* is used at every input pin in all Cypress products to provide protection against ESD. This circuitry has been designed to withstand repeated applications of high voltages without failure or performance degradation. This is accomplished by preventing the high (ESD) voltage from reaching the thin gate oxides of the internal transistors.

The circuit consists of two thick oxide (field) transistors wrapped around an input resistor (R_P) and a thin oxide (gate) transistor with a relatively low breakdown of 12 volts. Large input voltages cause the thick oxide transistors to turn on, discharging the ESD current to ground. The thin oxide transistor breaks down when the voltage across it (drain to source) exceeds 12 volts. It is protected from destruction by the current limiting action of R_P.

Experiments have confirmed that this input protection circuitry results in ESD protection in excess of 2001 volts.

Definition of Latchup

Latchup is a regenerative phenomenon that occurs when the voltage at an input pin or an output pin is either raised above the power supply voltage potential or lowered below the substrate voltage potential (which is usually ground).

Current rapidly increases until, in effect, a short circuit from V_{CC} to ground exists. If the (V_{CC}) current is not limited it will destroy the device; usually by melting a metal trace.

Causes of Latchup

The CMOS processing, which provides both N-channel and P-channel MOS transistors, also inherently provides parasitic bipolar transistors; both NPNs and PNPs. Latchup is caused when these parasitic transistors are inadvertently turned on.

As long as the voltages that are applied to the package pins of the CMOS IC remain within the limits of the power supply voltages (usually 0 volts to 5 volts), these parasitic bipolar transistors will remain dormant (i.e., off). However, when either negative voltages or positive voltages greater than the V_{CC} supply voltage are applied to input or output pins, these parasitic bipolar transistors may turn on and cause latchup.

Conditions For Latchup

A cross section of a typical CMOS inverter using a P-channel pullup transistor and an N-channel pulldown transistor is shown in *Figure 6.* Also shown is an N-channel output driver that is isolated from the CMOS inverter by a guard ring (channel stopper) that is necessary to prevent parasitic MOS transistors between devices. P+ guard rings surround N-channel devices and N+ guard rings surround P-channel devices. The parasitic SCR (PNPN) and bias generator are illustrated in *Figure 7.* The output driver schematic is not shown.

In order for latchup to occur two conditions must be satisfied; (1) the product of the betas of the NPN and PNP transistors must be greater than one, and (2) a trigger current must exist that turns on the SCR.

Since the SCR structure in bulk CMOS cannot be eliminated, preventing latchup is reduced to keeping the SCR from turning on. If either $R_{WELL} = 0$ or $R_{SUB} = 0$ the SCR cannot turn on because the base emitter junction of the PNP cannot be forward biased because they are tied together and the base emitter junction of the NPN cannot be forward biased because the base is connected to ground. Note, however, that the NPN could be turned on by a negative voltage on the output pin (if the right end of R_{SUB} is grounded).

Prevention of Latchup; Traditional Approaches

The traditional cures include increased horizontal spacing, diffused guard rings and metal straps to critical areas. These solutions are obviously opposite to the goal of greater density.

A brute-force approach that has been successful in reducing latchup has been to increase the conductivity of the N-well and the substrate. Changing the well

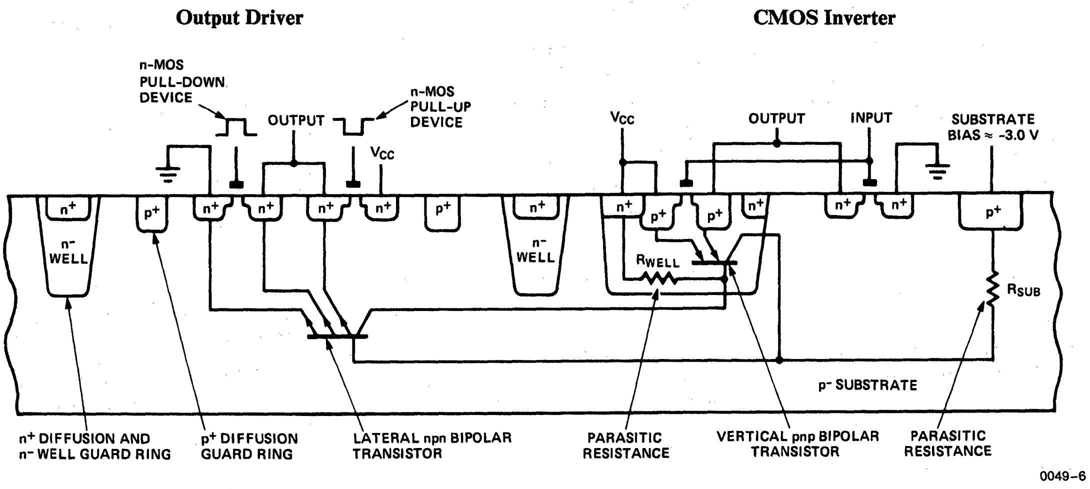
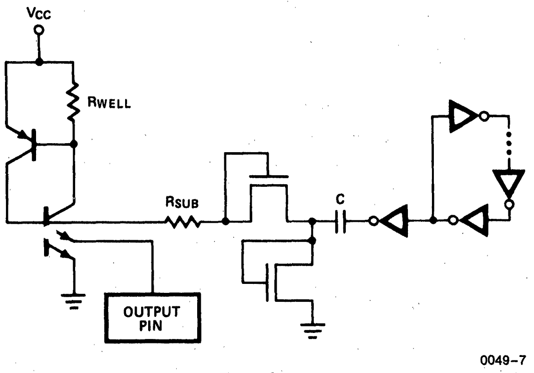


Figure 6. CMOS Cross Section and Parasitic Circuits

conductivity is unacceptable because it affects the characteristics of the P-channel MOS transistors. Using an epitaxial layer to reduce the substrate resistivity (R_{SUB}) is another possible solution.



Substrate Bias Generator →
Figure 7. Parasitic SCR and Bias Generator

The Cypress Solution to Latchup

Cypress uses several design techniques in addition to careful circuit layout and conservative design rules to eliminate latchup.

NMOS Output Pullup Transistors

Conventional CMOS technology uses a P-channel MOS as a pullup transistor on the output drivers. This has the advantage of being able to pull the output voltage HIGH level to within 100 millivolts of the positive voltage supply. However, this is of marginal value when TTL compatibility is required. In addition, the P-channel pullup is sensitive to overshoot and introduces another vertical PNP transistor that further compounds the latchup problem. Cypress uses N-channel pullup transistors that eliminate all of these problems and still maintain TTL compatibility.

Substrate Bias Generator

Cypress is the first company to use a substrate bias generator with CMOS technology. The bias generator keeps the substrate at approximately −3 volts DC, which serves several purposes.

Input Pins

The parasitic diodes shown in *Figure 5* cannot be forward biased unless the voltage at an input pin is at least one diode drop more negative than −3 volts. This translates into increased device tolerance to (negative voltage) undershoot at the input pins, caused by inductance in the leads. If the undershoot is this large, the output impedance of the bias generator itself is sufficient to prevent trigger current from being generated.

Output Pins

The same reasoning applies to negative voltages at the output pins as shown in *Figure 7.* In order to turn on the NPN transistor the voltage at the output pin must be at least one VBE more negative than −3 volts.

Guard Ring

To protect the "core" of the die from free floating holes and stray currents, a diffused collection guard ring that is strapped with metal and connected to the bias generator is used. This provides an effective wall against transient currents that could cause mis-reading of the EPROM cells.

CYPRESS SEMICONDUCTOR

PAL® C 16R6 Design Example: GCR Encoder/Decoder

Introduction

Digital encoding and decoding of data is often used to increase the reliability of data transmission and storage. One area where digital techniques are employed is the transformation between data stored on one-quarter inch magnetic tape and serial digital data.

This document describes the procedure used to encode/decode serial digital data for recording/reading from one-quarter inch magnetic tape using a Cypress CMOS PAL C 16R6 to implement the logic.

History

The recording format and the Group Code Recording (GCR) code have been adopted and incorporated in a series of standards by a committee called the QIC (Quarter Inch Cartridge) Committee, composed of manufacturers and users of quarter inch tapes and cartridges. The purpose of the committee is to insure compatibility between manufacturers and reliability to end users.

Quarter inch tape cartridges are used extensively to backup or archive data from hard disks. Most drives are operated in a continuous or streaming mode (for reasons that will be discussed later) and data is recorded at 10,000 FRPS (Flux Reversals Per Inch) in a serpentine manner on seven to fourteen channels. The tape moves at 30 to 90 ips (inches per second) and the error rates achieved are one in 10^9 or 10^{10}. A cartridge holds 2000 to 3000 feet of tape 0.001 inch thick and stores 20 to 80 million bytes (mega-bytes) of data.

Typical System

A block diagram of a typical system is shown in *Figure 1.* The interface between the Host (or Host Adapter) is bi-

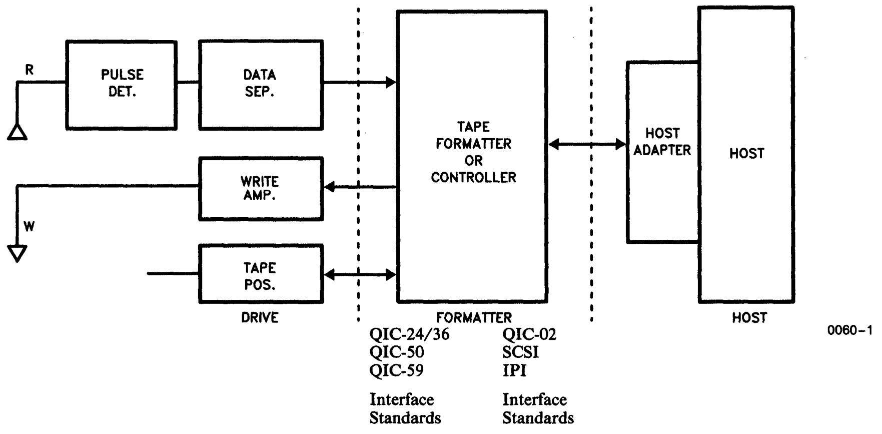


Figure 1. A Typical Tape Drive System

PAL® is a registered trademark of Monolithic Memories Inc.
ABEL™ is a trademark of Data I/O Corporation
PALASM™ is a trademark of Monolithic Memories Inc.
VAX™ is a trademark of Digital Equipment Corp.
WORDSTAR™ is a trademark of MicroPro International

Typical System (Continued)

directional, with a byte-wide data path and 10 to 20 control signals, depending upon the interface standard. Data rates are 300 KBs (thousand Bytes per second) to 1 MBs (Million Bytes per second).

The Formatter or Tape Controller performs serial/parallel conversion and encoding/decoding of the data as well as error checking and, in some cases, error correcting. Control is usually provided by a state machine that handles the handshaking between the host as well as control of the tape. Data is written in blocks of various lengths (depending upon the standard) and a "read after write" check is usually performed. Buffer storage of at least two blocks of data is usually provided using static RAMs (SRAMs), FIFOs, or some combination of the two.

The Drive electronics consist of digital signals that control and sense the tape motion and analog signals in the read and write paths. The interface between the Drive and the Formatter is digital and, once again, there are various standards.

Reading and Writing on Tape

To write on the tape a current of 100 mA or less is used to change the direction of magnetization. To read from the tape a coil of wire (the read head) is held against the tape and a voltage (10 mV or less) is induced by the change in direction of the magnetic flux on the tape.

Recording Codes

All codes used for recording on magnetic mediums are classified as Franaszek Run Length Limited (RLL) codes of the form:

(D, K)

where D = the minimum number of zeros between consecutive ones, and

K = the maximum number of zeros between consecutive ones.

D controls the highest frequency that can be recorded and K controls the lowest frequency.

Using the Franaszek notation, the GCR code is (1, 2). As illustrated in *Figure 2*, a flux reversal signifies a one and the absence of a flux reversal signifies a zero. This is true for all codes.

Peak Detection and Data Separation

Peaks are detected (versus zero crossings) because the circuits used are less sensitive to noise. The output of the peak detector goes to the most critical analog circuit in the drive; the data separator.

The function of the data separator is to provide ones and zeros that occur at a precise frequency. It does this by first synchronizing itself to a crystal controlled reference clock and then attempting to "lock" itself to the maximum data frequency on the tape by finding the phase difference between itself and the data output of the peak detector and driving a voltage controlled oscillator (VCO) such that they are equal. This is called a Phase Locked Loop (PLL). The frequency of the reference clock must be at least twice (2f) that of the highest frequency that is to be read (f).

The PLL is synchronized to the 2f reference frequency when it is not in use. A string of ones is recorded, which is called the preamble, before the block of data is recorded. When the command to read is given, the 2f reference frequency is removed from the data separator and the signal from the peak detector is applied to the data separator. The PLL then attempts to "lock" to the preamble. Just after the preamble, a code violation is recorded so that the Formatter can recognize where valid data begins. The procedure of locking onto the preamble is called "getting bit sync." The detection of the code violation is called "obtaining byte sync".

PLLs typically exhibit frequency and phase offsets during acquisition of the preamble. Phase errors also occur after lock, during the reading of the data field. Differences in tape speed during record and playback (as well as from unit to unit) result in frequency differences between the data read from the tape and the 2f reference.

Random phase errors caused by noise, intersymbol interference (bit crowding), timing errors and other transients may also get the PLL out of lock.

The data separator's PLL is susceptible to these errors because it must satisfy two conflicting conditions: (1) it must

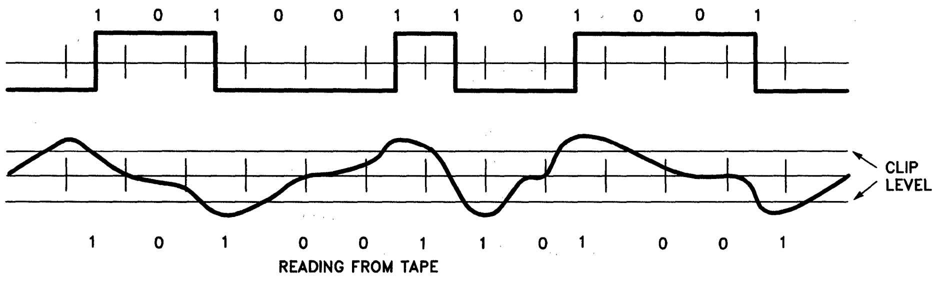


Figure 2

Reading and Writing on Tape (Continued)

lock quickly enough to detect the preamble, but (2) it must not overcorrect phase for a single misaligned bit.

Strings of zeros cause the phase of the PLL to shift and if the shift is larger than the "bit window", an error will occur. The QIC-24 standard calls for up to 37% bit shift tolerance, which means that the data separator must be able to recognize a "one" (flux transversal) that deviates ± 18.5% from its expected time position without causing a data error. In order to achieve this performance a four-bit binary nibble is encoded into a five-bit "GCR code word" that is written onto the tape.

Reasons for the GCR Code

The 5-bit GCR code format is required to encode the data such that no more than two consecutive zeros occur in the serial data. This encoding relaxes the performance requirements of the PLL and the loop filter so that the desired system performance can be achieved.

Static Tolerances

Another reason for GCR encoding is to compensate for the speed variation of the tape due to:

Mechanical Tolerances
  Cartridge
  Tape thickness (± 3%)

Tape Elasticity and Wear

Motor Speed Variation

Temperature and Humidity

The preceding static tolerances can result in a ± 10% speed variation of the tape.

Dynamic Tolerances

In addition to the static tolerances, there are Instantaneous Speed Variations (ISV) due to discontinuous tape release at the unwind spool (10–20%), guide/back stick slip (5%) and shuffle ISV (vibration) due to start/stop (5–30%). The shuffle ISV can be avoided by operating the tape in a continuous (streaming) mode. If these dynamic tolerances are added together they can result in a ± 15% speed variation.

Electronics Compensate

The electronics in the tape controller and the drive are designed to compensate for the tape speed variations due to the mechanical tolerances.

The compensation is performed by:

Data Encoding and Error Detection and Correction

Phase Locked Loop Design

Bit Window Tolerance

Sequence of Operations

During a write operation the following sequence occurs:

1. Idle (Hold)
2. Convert 4-bit parallel input to 5-bit GCR code and load into 5-bit register.
3. Shift out 5-bits to write amplifier.

During a read operation the following sequence occurs:

1. Idle (same as during write)
2. Shift in 5-bits.
3. Detect sync mark

   Set/Clear invalid flag

   Convert 5-bit serial input to 4-bit binary value and load into register.

Note: that the read clock and the write clock are not the same.

Also, the logic must keep up with the tape data rate.

And finally, the read and write operations are mutually exclusive so that the storage elements (D flip-flops) can be time-shared and that read and write operations require 5 clocks.

A total of 5 states are required because the idle state is common to both read and write operations. Therefore, 3 control lines will be required. It is convenient to designate one control line as an enable line (active LOW) and the other two lines as Mode Control signals.

The control of these lines is not described here, nor is the required clock synchronization. The reason for not doing this is that at the next level of control, system considerations such as what action to take when errors occur must be implemented in hardware and these tend to be not only application dependent but also very subjective.

The diagrams of *Figure 3* show the flow of data under the control of the ENABLE signal and the M0 and M1 mode control signals.

The GCR Code

The GCR code is part of the QIC-24 Standard and is also the ANSI X3.54 standard (1976). The MSB (leftmost bit) is recorded first. Note that there are a maximum of two consecutive zeros in the five-bit code that is recorded on the tape.

	4-Bit Code				5-Bit Code				
Line Number (For Ref.)	**D 3**	**D 2**	**D 1**	**D 0**	**Y 3**	**Y 2**	**Y 1**	**Y 0**	**S o**
0	0	0	0	0	1	1	0	0	1
1	0	0	0	1	1	1	0	1	1
2	0	0	1	0	1	0	0	1	0
3	0	0	1	1	1	0	0	1	1
4	0	1	0	0	1	1	1	0	1
5	0	1	0	1	1	0	1	0	1
6	0	1	1	0	1	0	1	1	0
7	0	1	1	1	1	0	1	1	1
8	1	0	0	0	1	1	0	1	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	0	1	0	1	0
11	1	0	1	1	0	1	0	1	1
12	1	1	0	0	1	1	1	1	0
13	1	1	0	1	0	1	1	0	1
14	1	1	1	0	0	1	1	1	0
15	1	1	1	1	0	1	1	1	1
	A 3	A 2	A 1	A 0	B 0	B 1	B 2	B 3	B 4

Figure 4. GCR Code

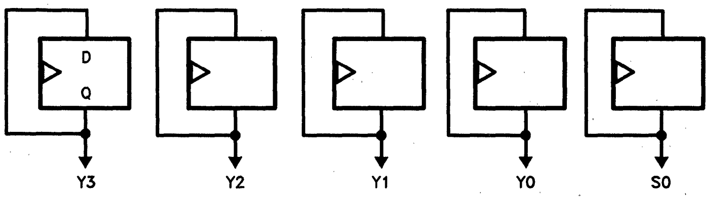
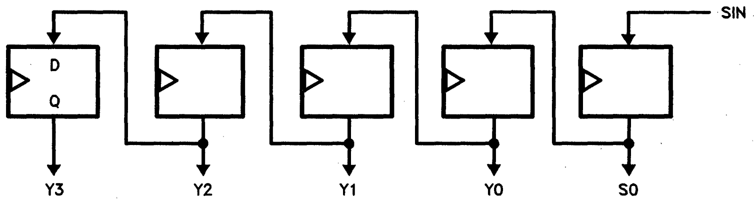
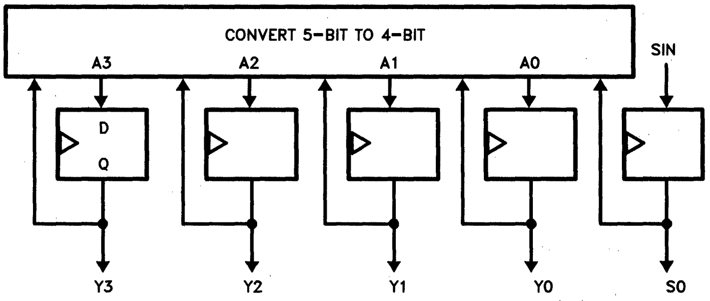
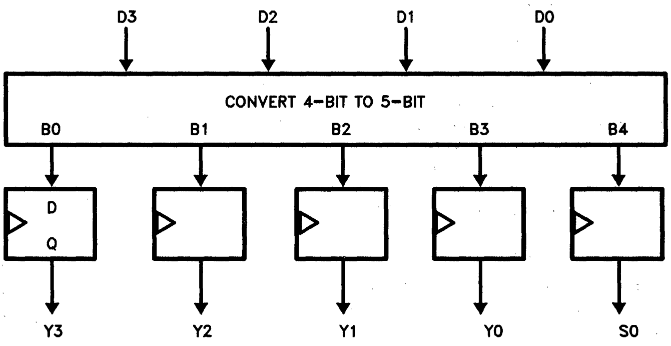
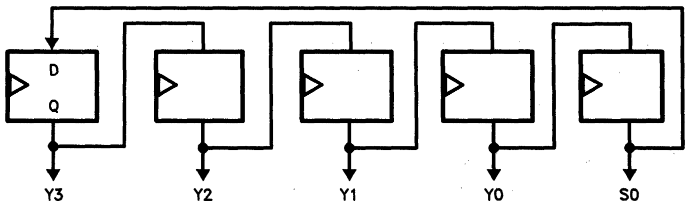
ENABLE	M1	M0	OPERATION	DATA FLOW DIAGRAM
1	X	X	HOLD	
0	0	0	SERIAL SHIFT IN	
0	1	0	CONVERT 5-BIT TO 4-BIT	
0	1	1	CONVERT 4-BIT TO 5-BIT	
0	0	1	SERIAL SHIFT OUT	

Figure 3. Data Flow Diagrams

Design Procedure

The design procedure will be to map the code conversions using Venn diagrams and write the logic equations as the "sum of products" or in minterm form. Six flip-flops are required, so the logic will be implemented using a PAL C 16R6. Because the PAL device has inverting output buffers, the zeros will be mapped. The D flip-flops require an "extra term" for them to hold their states when the ENABLE is HIGH.

For example, for a conventional D flip-flop the form of the logic equations would be:

```
D =    ENABLE 1 ( Q )     ; RECIRCULATE
                            PRESENT
                            STATE
   +   ENABLE 2 ( F2 )    ; FUNCTION 2
   +   ENABLE 3 ( F3 )    ; FUNCTION 3
```

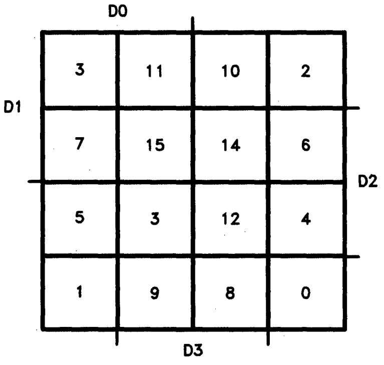
Where the ENABLE controls are mutually exclusive.

4-Bit to 5-Bit Conversion for Y3 Output

In *Figure 4* (at the bottom) the 5-bit code columns are labeled B0 through B4 to help the reader understand how the 4-bit code is mapped. In addition, the line numbers are labeled 0 through 15, which correspond to the values of the 4-bit binary code.

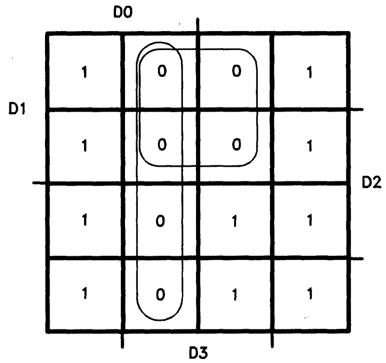
Figure 5a shows how the 4-bit binary code is mapped on the Venn diagram. For example, reference line number zero, which corresponds to binary value zero, is located in the lower right hand corner of *Figure 5a.*

The Venn diagram of *Figure 5b* shows the conversion for the $\overline{Y3}$ output. It is labeled the B0 input to the D flip-flop. Note that the parallel nibble (see *Figure 3*) is reversed (end for end) so that the MSB is written first when it is shifted out.



0060-4

Figure 5a. Binary Values



0060-5

Figure 5b. $\overline{Y3}$ Map

In *Figure 5b,* the ones and zeros in column B0 are mapped. For example, reference line zero has the value 1 in column B0 of *Figure 4.* Therefore, a one is placed in the square corresponding to binary value zero in *Figure 5b.* In a similar manner, ref. line 15 has a value of zero in column B0, so a zero is placed in the square corresponding to binary value fifteen.

Writing the Equation

If the output of the PAL C 16R6 were positive true logic, we would write the equation to include all of the ones on the Venn diagram. However, because the PAL device output is negative logic (active LOW) we will write the equation to include all of the zeros. Then, when the PAL device inverts the signals, the zeros will be changed to ones, so that the final outputs will be positive true logic.

By inspection:

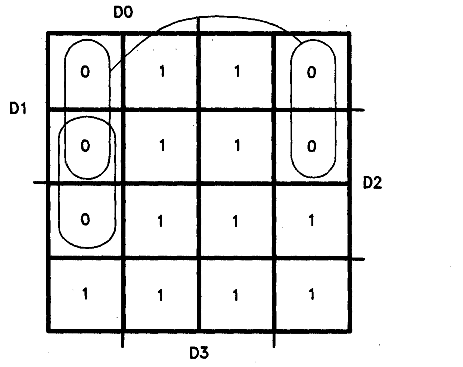
$\overline{B0}$ = D3 D0 + D3 D1 or,

$\overline{Y3}$ = D3 D0 + D3 D1

Design Procedure (Continued)

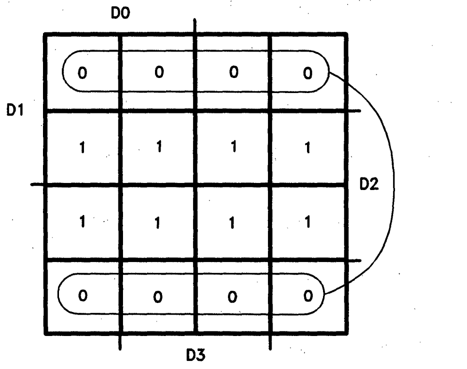
4-Bit to 5-Bit Conversions for $\overline{Y2}$, $\overline{Y1}$, $\overline{Y0}$, $\overline{So}$

These are presented for the sake of completeness.



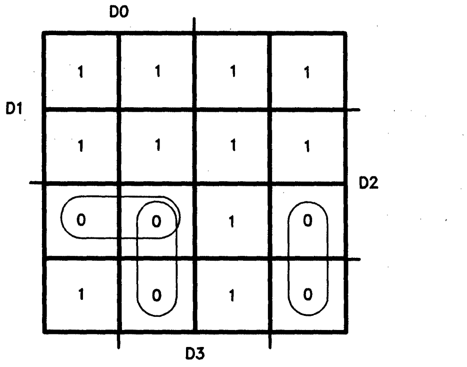
$\overline{Y2} = \overline{B1} = \overline{D3}\,D1 + \overline{D3}\,D2\,D0$

Figure 5c. $\overline{Y2}$ Map



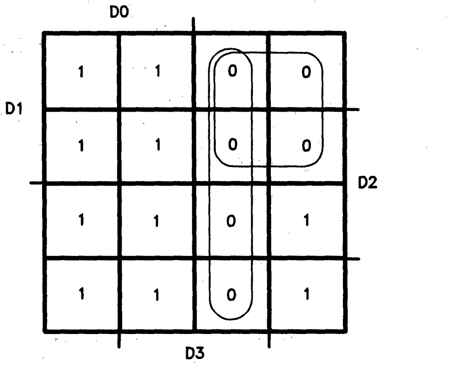
$\overline{Y1} = \overline{B2} = \overline{D2}$

Figure 5d. $\overline{Y1}$ Map



$\overline{Y0} = \overline{B3} = \overline{D3}\,\overline{D1}\,\overline{D0} + D3\,\overline{D1}\,D0 + D2\,\overline{D1}\,D0$

Figure 5e. $\overline{Y0}$ Map



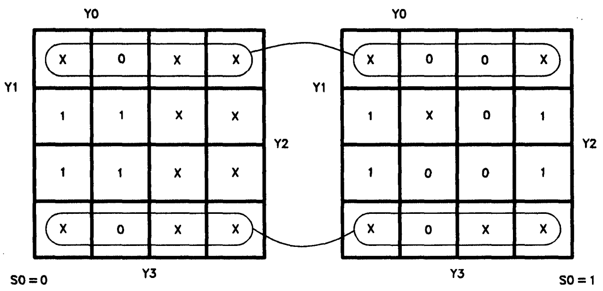
$\overline{So} = \overline{B4} = D1\,\overline{D0} + D3\,\overline{D0}$

Figure 5f. $\overline{So}$ Map

5-Bit to 4-Bit Conversion for $\overline{Y}$ Outputs

This conversion requires two 16 square Venn diagrams because there are $2^5 = 32$ possible binary values. However, note that in *Figure 4* not all 32 possible combinations are used in the 5-bit code columns. These unused combinations are "don't cares", which are represented by Xs in the Venn diagrams, which can be either ones or zeros, which further reduces or simplifies the logic equations.

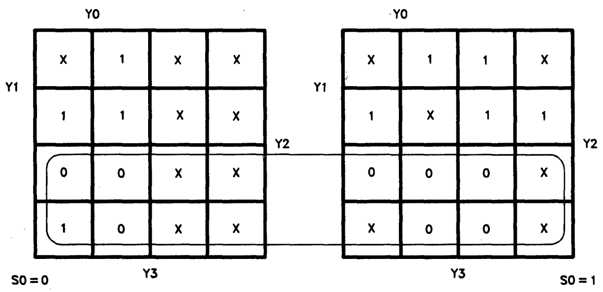
The procedure is: plot the 1s and 0s
put Xs in the blank squares
write the equations for the zeros.



0060-10

$\overline{Y3} = \overline{A3} = \overline{Y2} + Y3\ So$

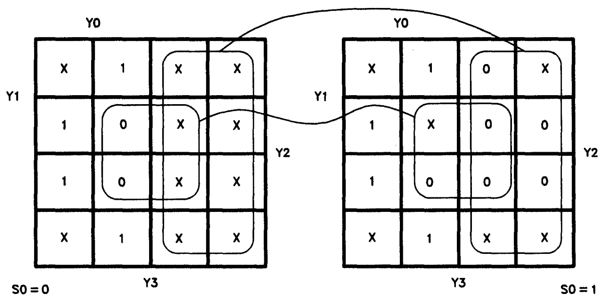
Figure 6a



0060-11

$\overline{Y2} = \overline{A2} = \overline{Y1}$

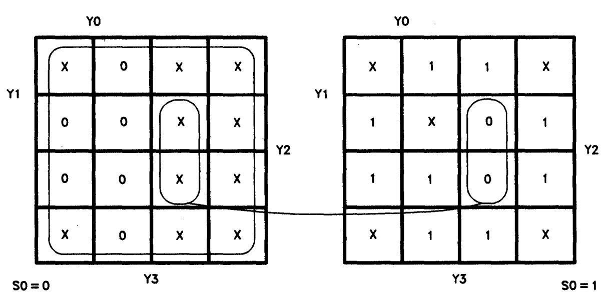
Figure 6b



0060-12

$\overline{Y1} = \overline{A1} = \overline{Y0} + Y3\ Y2$

Figure 6c



0060-13

$\overline{Y0} = \overline{A0} = Y3\ Y2\ \overline{Y0} + \overline{So}$

Figure 6d

Design Procedure (Continued)

Serial Shift In

During serial shift in (both mode control signals LOW) the data output of the data separator is applied to the input of the formatter. The signal is called SIN and is applied to the D input of the SOUT flip-flop. The output of the SOUT flip-flop is applied to the D input of the Y0 flip-flop and its output is applied to the input of the Y1 flip-flop, etc. After five read clocks the MSB of the 5-bit GCR coded data is in Y3 and the LSB is in SOUT.

Serial Shift Out

During a write operation, after the 4-bit data is converted to 5-bit data and reversed, it is shifted out using the write clock and written on tape. The shift direction is opposite to that in Serial Shift In. Note that it is right shifted "end around" (see *Figure 3*) so that after 5 write clocks the same data appears in the register.

Invalid Flag (INV Flip-Flop)

The Invalid flip-flop is set to a one when an invalid 5-bit code is read from the tape. This is used to tell the tape Formatter that the next data read is the beginning of the data block. This procedure is called getting "byte sync." INV is a negative true signal, so the logic equations are written for ones on the Venn diagram.

The 16 binary values that are NOT listed in *Figure 4* are plotted as ones in *Figure 7*. The procedure was to plot zeros in the squares where there were valid 5-bit codes, then fill the rest with ones and then write the equation for the ones.

The Invalid flip-flop is enabled by a signal called CIF (Control Invalid Flag) and reset when CIF is LOW.

Synchronization Mark Detection

Bit synchronization is achieved when the illegal 5-bit code of all ones is read from the tape. It is the logical AND of all five bits, or BS = Y3 • Y2 • Y1 • Y0 • SOUT.

Implementation Procedure

Once the conceptual design has been completed, it must be reduced to practice. There are two main steps in the process;

1. describe the logic using a high-level language, and
2. program the PAL device.

Several programs that run on the IBM PC (or equivalent) or the VAX™ computer are available from either semiconductor manufacturers or from third party software vendors. The first such program, called PALASM™ (PAL device Assembler) was developed by Monolithic Memories. It enables the designer to describe the logic in terms of Boolean equations, truth tables, or state diagrams using a language whose syntax is comparable to a microcomputer assembly language.

PALASM Equations

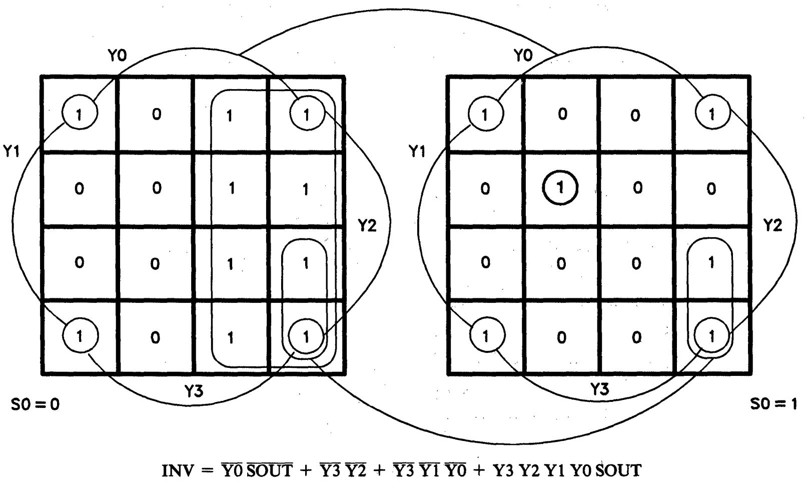
The equations were written in the PALASM syntax. The (ASCII) file created using WORDSTAR in the non-document (N) mode is shown in *Figure 8*.

Conversion to ABEL™

The PALASM file (GCREX.PAL) was then translated to ABEL syntax using the TOABEL program. The format of the command is:

```
TOABEL -IB:GCREX -OB:GCREXT
```

The TOABEL program converted the GCREX.PAL file to a file named GCREXT.ABL, whose listing is shown in *Figure 9*.



$$INV = \overline{Y0}\,\overline{SOUT} + \overline{Y3}\,\overline{Y2} + \overline{Y3}\,\overline{Y1}\,\overline{Y0} + Y3\,Y2\,Y1\,Y0\,SOUT$$

Figure 7

ABEL Program Procedure

The ABEL program consists of an executive and several overlay programs that are executed by simply typing in;

ABEL B:GCREXT

followed by an enter (CR) from the keyboard of an IBM (or look-alike) PC. The ABEL program was developed by a programmer manufacturer, Data I/O Corporation. The source file may be simplified (logic reduction), a logic simulation may be performed, and test vectors may be generated.

ABEL Programs

The ABEL programs are:

Program Name	Function
PARSE	Read source file, check syntax, expand macros, act upon assembler directives.
TRANSFOR	Convert the description to an intermediate form.
REDUCE	Perform logic reduction.
FUSEMAP	Create the programmer load (JEDEC) file.
SIMULATE	Simulate the operation of a programmed device.
DOCUMENT	Create a design documentation file.

ABEL Outputs

The output files are:

GCREXT.LST	
GCREXT.OUT	
GCREXT.DOC	see *Figure 10*
GCREXT.SIM	(This design was not simulated.)
P16R6.JED	see *Figure 11*

The last file is in JEDEC (JC-42.1-81-62) format; suitable for loading into a PLD programmer. The listing is shown in *Figure 11.* The DOCUMENT program output is shown in *Figure 10.*

Programming the 16R6

The 16R6 was programmed using the Data I/O model 29B programmer operated in the remote mode to the PC. The design was then verified by checking out the device on the bench.

Summary

Space Saving Advantage

This design example illustrates the space saving advantage of Cypress CMOS PAL devices. The FUSEMAP program printed out that 40 of the 64 available product terms were used.

If the PALASM input equations of *Figure 8* are implemented in two-input gates, approximately thirty gates are required for each one of the six D flip-flop inputs, or a total of 6 × 30 = 180 two-input gates. The logic equations alone would then require 180 divided by 4 = 45 14 pin DIPs. The six flip-flops would require three 14 pin DIPs for a total of 48 DIPs. This example demonstrates the power of the Cypress PAL devices.

Power Saving Advantage

The maximum I_{CC} current, under worst case conditions, for the PAL C 16R6L-25PC is 45 mA.

If the typical I_{CC} per package is assumed to be 10 mA, the total I_{CC} for 50 TTL packages would be 500 mA.

The worst case I_{CC} for the TTL system could be as high as 20 mA per DIP, which would mean a total of one Ampere for the system.

The Cypress CMOS PAL device results in a system power reduction of between a factor of 10 or 15, depending upon whether typical or worst case numbers are compared.

PALASM Equations

```
PAL16R6                DESIGN EXAMPLE                   FILENAME; GCREX.PAL
PAT001                                                  BRUCE WENNIGER 9/17/85
4B-5B ENCODER/DECODER
CYPRESS SEMICONDUCTOR
 CK M1  M0   D3 D2 D1 D0 /EN  /CIF GND
/E  SIN /INV Y0 Y1 Y2 Y3 SOUT /BS  VCC
/SOUT:= EN*/SOUT                                        +         ; HOLD/RECIRCULATE
        /EN*/M1*/M0*/SIN                                +         ; SERIAL SHIFT IN
        /EN*/M1*M0*/Y0                                  +         ; SERIAL SHIFT OUT
        /EN*/M1*/M0*/SIN                                +         ; CONV. SIN & LOAD
        /EN*/M1* M0* D1*/D0                             +         ; CONV. PAR. & LOAD
        /EN*/M1* M0* D3*/D0                                       ; DITTO
```

Figure 8

PALASM Equations (Continued)

```
/Y0     := EN*/Y0                                  +     ; HOLD
           /EN*/M1*/M0*/SOUT                       +     ; SERIAL SHIFT IN
           /EN*/M1* M0*/Y1                         +     ; SERIAL SHIFT OUT
           /EN* M1*/M0*/SOUT                       +     ; CONV. SIN & LOAD
           /EN* M1*/M0* Y3* Y2*/Y0                 +     ; DITTO
           /EN* M1* M0* D2*/D1*D0                  +     ; CONV. PAR. & LOAD
           /EN* M1* M0* D3*/D1* D0                 +     ; DITTO
           /EN* M1* M0*/D3*/D1*/D0                       ; DITTO

/Y1     := EN*/Y1                                  +     ; HOLD
           /EN*/M1*/M0*/Y0                         +     ; SERIAL SHIFT IN
           /EN*/M1* M0*/Y2                         +     ; SERIAL SHIFT OUT
           /EN* M1*/M0*/Y0                         +     ; CONV. SIN & LOAD
           /EN* M1*/M0* Y3* Y2                     +     ; DITTO
           /EN* M1* M0*/D2                               ; CONV. PAR. & LOAD

/Y2     := EN*/Y2                                  +     ; HOLD
           /EN*/M1*/M0*/Y1                         +     ; SERIAL SHIFT IN
           /EN*/M1* M0*/Y3                         +     ; SERIAL SHIFT OUT
           /EN* M1*/M0*/Y1                         +     ; CONV. SIN & LOAD
           /EN* M1* M0*/D3* D1                     +     ; CONV. PAR. & LOAD
           /EN* M1* M0*/D3* D2* D0                       ; DITTO

/Y3     := EN*/Y3                                  +     ; HOLD
           /EN*/M1*/M0*/Y2                         +     ; SERIAL SHIFT IN
           /EN*/M1* M0*/SOUT                       +     ; SERIAL SHIFT OUT
           /EN* M1*/M0* Y3* SOUT                   +     ; CONV. SIN & LOAD
           /EN* M1*/M0*/Y2                         +     ; DITTO
           /EN* M1* M0* D3* D0                     +     ; CONV. PAR. & LOAD
           /EN* M1* M0* D3* D1                           ; DITTO

INV     :=/CIF* INV                                +     ; HOLD INV FLAG
                                                         ; (ACTIVE LOW)
           CIF* M1*/M0*/Y3*/Y2                     +     ; SET IF INVALID
           CIF* M1*/M0*/Y3/Y1*/Y0                  +     ; DITTO
           CIF* M1*/M0*/Y0*/SOUT                   +     ; DITTO
           CIF* M1*/M0* Y3* Y2* Y1* Y0* SOUT       +     ; DITTO

BS      = Y3* Y2* Y1* Y0* SOUT                           ; BIT SYNC.
                                                         ; (ACTIVE LOW)
```

Figure 8 (Continued)

ABEL Listing

```
module --gcrext;                    flag '-r0;
title
'PAL16R6                            DESIGN EXAMPLE                    FILENAME: GCREX.PAL
PAT001                                           BRUCE WENNIGER 9/17/85
4B-5B ENCODER/DECODER
CYPRESS SEMICONDUCTOR
-Translated by TOABEL-';
P16R6 device 'P16R6';


"declarations
             TRUE,FALSE = 1,0;
             H,L = 1,0;
             X,Z,C = .X.,.Z.,.C.;
             GND,VCC
                     pin   10,20;
             CK,M1,M0,D3,D2,D1,D0,EN,CIF,E
                     pin   1,2,3,4,5,6,7,8,9,11;
             INV,Y0,Y1,Y2,Y3,SOUT
                     pin   13,14,15,16,17,18;
             SIN,BS
                     pin   12,19;
equations
             !SOUT    := !EN & !SOUT
                     # EN & !M1 & !M0 & !SIN
                     # EN & !M1 & M0 & !Y0
                     # EN & M1 & !M0 & !SIN
                     # EN & M1 & M0 & D1 & !D0
                     # EN & M1 & M0 & D3 & !D0 ;
             " HOLD/RECIRCULATE
             " SERIAL SHIFT IN
             " SERIAL SHIFT OUT
             " CONV. SIN & LOAD
             " CONV. PAR. & LOAD
             " DITTO
             !Y0      := !EN & !Y0
                     # EN & !M1 & !M0 & !SOUT
                     # EN & !M1 & M0 & !Y1
                     # EN & M1 & !M0 & !SOUT
                     # EN & M1 & !M0 & Y3 & Y2 & !Y0
                     # EN & M1 & M0 & D2 & !D1 & D0
                     # EN & M1 & M0 & D3 & !D1 & D0
                     # EN & M1 & M0 & !D3 & !D1 & !D0;
```

Figure 9

ABEL Listing (Continued)

```
" HOLD
" SERIAL SHIFT IN
" SERIAL SHIFT OUT
" CONV. SIN & LOAD
"DITTO
"CONV. PAR. & LOAD
"DITTO
"DITTO

!Y1         := !EN & !Y1
             # EN & !M1 & !MO & !YO
             # EN & !M1 & MO & !Y2
             # EN & M1 & !MO & !YO
             # EN & M1 & !MO & Y3 & Y2
             # EN & M1 & MO & !D2 ;

"HOLD
"SERIAL SHIFT IN
"SERIAL SHIFT OUT
"CONV. SIN & LOAD
"DITTO
"CONV. PAR. & LOAD

!Y2         := !EN & !Y2
             # EN & !M1 & !MO & !Y1
             # EN & !M1 & MO & !Y3
             # EN & M1 & !MO & !Y1
             # EN & M1 & MO & !D3 & D1
             # EN & M1 & MO & !D3 & D2 & DO

"HOLD
"SERIAL SHIFT IN
"SERIAL SHIFT OUT
"CONV. SIN & LOAD
"CONV. PAR. & LOAD
"DITTO

!Y3         := !EN & !Y3
             # EN & !M1 & !MO & !Y2
             # EN & !M1 & MO & !SOUT
             # EN & M1 & !MO & Y3 & SOUT
             # EN & M1 & !MO & !Y2
             # EN & M1 & MO & D3 & DO
             # EN & M1 & MO & D3 & D1 ;
```

Figure 9 (Continued)

ABEL Listing (Continued)

```
            "HOLD
            "SERIAL SHIFT IN
            "SERIAL SHIFT OUT
            "CONV. SIN & LOAD
            "DITTO
            "CONV. PAR. & LOAD
            "DITTO

            !INV        := CIF & !INV
                        # !CIF & M1 & !M0 & !Y3 & !Y2
                        # !CIF & M1 & !M0 & !Y3 & !Y1 & !Y0
                        # !CIF & M1 & !M0 & !Y0 & !SOUT
                        # !CIF & M1 & !M0 & Y3 & Y2 & Y1 & Y0 & SOUT ;
            " HOLD INV FLAG
            " SET IF INVALID
            " DITTO
            " DITTO
            " DITTO

            !BS         = Y3 & Y2 & Y1 & Y0 & SOUT;
            " BIT SYNC.
end --gcrext;
```

Figure 9 (Continued)

Document File

```
                                                          Page 1
ABEL™ Version 1.10 - Document Generator                   17-Sept-85 8:30 AM
PAL16R6             DESIGN EXAMPLE              FILENAME; GCREX.PAL
PAT001                           BRUCE WENNIGER 9/17/85
4B-5B ENCODER/DECODER
CYPRESS SEMICONDUCTOR
-Translated by TOABEL-
Equations for Module --gcrext
Device P16R6
        Reduced Equations:
           SOUT := !( !EN & !SOUT
                   # EN & !MO & !M1 & !SIN
                   # EN & MO & !M1 & !YO
                   # EN & !MO & M1 & !SIN
                   #  !DO & D1 & EN & MO & M1
                   #  !DO & D3 & EN & MO & M1) ;
           YO :=   !( !EN & !YO
                   # EN & !MO & !M1 & !SOUT
                   # EN & MO & !M1 & !Y1
                   # EN & !MO & M1 & !SOUT
                   # EN & !MO & M1 & !YO & Y2 & Y3
                   # DO & !D1 & D2 & EN & MO & M1
                   # DO & !D1 & D3 & EN & MO & M1
                   #  !DO & !D1 & !D3 & EN & MO & M1) ;
           Y1 :=   !( !EN & !Y1
                   # EN & !MO & !M1 & !YO
                   # EN & MO & !M1 & !Y2
                   # EN & !MO & M1 & !YO
                   # EN & !MO & M1 & Y2 & Y3
                   #  !D2 & EN & MO & M1) ;
           Y2 :=   !( !EN & !Y2
                   # EN & !MO & !M1 & !Y1
                   # EN & MO & !M1 & !Y3
                   # EN & !MO & M1 & !Y1
                   # D1 & !D3 & EN & MO & M1
                   # DO & D2 & !D3 & EN & MO & M1) ;
           Y3 :=   !( !EN & !Y3
                   # EN & !MO & !M1 & !Y2
                   # EN & MO & !M1 & !SOUT
                   # EN & !MO & M1 & SOUT & Y3
                   # EN & !MO & M1 & !Y2
                   # DO & D3 & EN & MO & M1
                   # D1 & D3 & EN & MO & M1) ;
           INV =   !(CIF & !INV
```

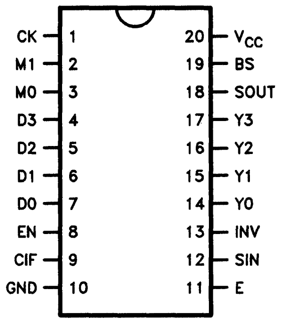
Figure 10

Document File (Continued)

```
                                                         Page 1
ABEL™ VERSION 1.10 - Document Generator                  17 Sept-85 8:30 AM
PAL16R6                            DESIGN EXAMPLE            FILENAME: GCREX.PAL
PAT001                                                 BRUCE WENNIGER 9/17/85
4B-5B ENCODER/DECODER
CYPRESS SEMICONDUCTOR
-Translated by TOABEL-
Equations for Module --gcrext
Device P16R6
              # !CIF & !MO & M1 & !Y2 & !Y3
              # !CIF & !MO & M1 & !YO & !Y1 & !Y3
              # !CIF & !MO & M1 & !SOUT & !YO
              # !CIF & !MO & M1 & SOUT & YO & Y1 & Y2 & Y3);
       BS = !(SOUT & YO & Y1 & Y2 & Y3);
Chip diagram for Module --gcrext
Device P16R6
```

Figure 10 (Continued)

PAL C 16R6



0060-15

```
end of module --gcrext
```

JEDEC File

```
ABEL™ Version 1.10 JEDEC file for: P16R6
Created on: 17-Sept-85 8:30 AM
PAL16R6              DESIGN EXAMPLE              FILENAME: GCREX.PAL
PAT001                                   BRUCE WENNIGER 9/17/85
4B-5B ENCODER/DECODER
CYPRESS SEMICONDUCTOR
-Translated by TOABEL-*
QP20* QF2048*
L0000
11111111111111111111111111111111
11111101110111011101110111111111
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
11111110111111111111111110111111
10111011111111111111111101111110
10110111111111111111111001111111
01111011111111111111111101111110
01110111111111110111101101111111
01110111011111111111101101111111
00000000000000000000000000000000
00000000000000000000000000000000
11111111111011111111111110111111
10111011111111101111111101111111
10110110111111111111111101111111
01111001110111111111111101111111
01111011111111101111111101111111
01110111011111111111011101111111
01110111011111110111111101111111
00000000000000000000000000000000
00000000000000000000000000000000
11111111111011111111111110111111
10111011111111101111111101111111
10110110111111111111111101111111
01111001110111111111111101111111
01111011111111101111111101111111
01111111111111111111011101111111
01110111011111110111111101111111
00000000000000000000000000000000
11111111111111101111111110111111
```

Figure 11

JEDEC File (Continued)

```
10111011111111111110111101111111
10110111111011111111111101111111
01111011111111111110111101111111
01110111101111110111111101111111
01110111101101111111011101111111
00000000000000000000000000000000
00000000000000000000000000000000
11111111111111111110111101111111
10111011111111111111111001111111
10110111111111101111111101111111
01111011111111111111111001111111
01111011110111011111111101111111
01110111111110111111111101111111
00000000000000000000000000000000
00000000000000000000000000000000
11111111111111111111111010111111
10111010111111111111111101111111
10110111111111111110111101111111
01111011111111111111111101111111
01111011110111011111111001111111
01110111111101111011011101111111
01110111011111111011011101111111
01110111101111111011101101111111
11111111111111111111111111100111
01111011111011101111111111111011
01111011111011111110110111111011
01111010111111111111111011111011
01111001110111011101110111111011
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000*
C8E51*
D15A
```

Figure 11 (Continued)

NOTES:

CYPRESS SEMICONDUCTOR

Using ABEL to Program the Cypress 22V10

Introduction

This document is a compilation of application examples using the popular PALC22V10 Programmable Logic Device. The examples have been chosen to demonstrate the advanced features of the PALC22V10 and some of the high-level logic description techniques of the ABEL programming langauge. Each of the first seven examples illustrates a specific PALC22V10 feature. The feature is described and the ABEL programming language statements necessary to implement the feature are listed. The ABEL files contain test vectors that exercise the feature. The remaining examples are complete PALC22V10 designs that combine many of the individual features. All of the examples have been tested and are available, by request, on floppy disk from Cypress Semiconductor. The design examples provided are:

1. Asynchronous Reset/Synchronous Preset from Single Inputs
2. Asynchronous Reset/Synchronous Preset from Product Terms
3. Asynchronous Reset/Synchronous Preset Used to Load Predetermined Non-Zero Values using 'Istype' Statements
4. Output Enable Control from a Single Input
5. Output Enable Control from Product Terms
6. Using 16 Product Terms - An 8-bit Identity Comparitor
7. Using Feedback to Realize More than 16 Product Terms in a 9-bit Single Output Identity Comparitor
8. Bi-Directional I/O - Bus Interface with Answer-back
9. 10-bit Address Generator / Multiplexer
10. Triple State Machine Example

The PALC22V10 application examples are meant to be used as a reference for design engineers. These are excellent tools both for the designer new to programmable logic and for the veteran PLD user. All are encouraged to add the files to their ABEL source file libraries and to include any part of the files in their own designs. The files may be used as a template by editing them using any text editor in the non-document mode. Conversion to the CUPL® or PLD ToolKit® programming language is easily accomplished; the syntactical similarity of these languages makes this possible. For conversion to other languages, consult your user's guide.

Notes on the ABEL Programming Language

This section is provided as a brief introduction to the structure and syntax of the ABEL programming language. A rudimentary understanding of the ABEL language is necessary to fully appreciate the example files included in this brief. Experienced ABEL users may skip this section and proceed directly to the examples.

An ABEL source file provides the information necessary to describe the logical operation of a PLD design. The keywords and structure of these files can be seen in any of the examples. Source files are processed by the ABEL language processor which yields a JEDEC programming file and documentation of the design. The language processor also uses test vectors generated by the designer as part of the source file to test the functionality of the design.

ABEL Design Entry Methods

The ABEL programming language offers three methods for defining the logical operation of given design. These methods are:

1. Boolean Equation
2. Truth Table
3. State Diagram

A source file may include any or all of these design entry methods. The following sections describe the Boolean equation, truth table, and state diagram entry methods as well as the operators and notation conventions used in the source files.

ABEL Operators and Notation Conventions

In addition to the standard AND and OR logical operators, ABEL supports several high-level logic definitions. "+" and "*" signs, which in standard Boolean notation stand for OR and AND operations respectively, are interpreted by ABEL to be arithmetic addition and multiplication. This greatly simplifies the design of counter and ALU logic. *Table 1* below shows the logical operators supported by ABEL. The labels A, B, and C in the examples may be either individual pins or a set of pins as defined in the source file.

Table 1. ABEL Logical Operators

Operator	Definition	Example
!	NOT: ones compliment	C = !A;
&	AND	C = A & B;
#	OR	C = A # B;
$	XOR: exclusive OR	C = A $ B;
!$	XNOR: exclusive NOR	C = A !$ B;

Note that these operators may be used with operands of more than one bit on a bit by bit basis. For example, the result of logically ORing hexidecimal values of 8 and 2 yields hexidecimal value A:

^h08 # ^h02 = ^h0A

Specifying Alternate Number Bases

Note the "^**h**" symbols in the example above. This symbol instructs the language processor to interpret the value following the symbol as base-16 (hex). The default number base in ABEL is decimal but can be changed for individual expressions with ^**b** (for binary), ^**o** (for octal), ^**d** (for decimal), or ^**h** (for hexidecimal). The "@ **radix**" command can be used to change the default number base to binary, octal, decimal or hexidecimal for all subsequent statements in a source document. The command "@ radix 16" is used in all of the source files in this brief to set the number base to hexidecimal.

Arithmetic Operators

Arithmetic operators are provided to allow for easy implementation of math and shifting functions. *Table 2* lists the arithmetic operators supported by ABEL.

Table 2. ABEL Arithmetic Operators

Operator	Definition	Example
-	2s complement	C = -A;
-	subtraction	C = A - B;
+	addition	C = A + B;
*	multiplication	C = A * B;
/	integer division	C = A / B;
%	remainder	C = A % B;
<	shift left	C = A < 2; (shift left 2 bits)
>	shift right	C = A > 3; (shift right 3 bits)

Shifting operations are unsigned and zeros are shifted into the side of the expression opposite the direction of the shift. Also note that the symbol "/" is interpreted as an unsigned division operation. Other programmable logic languages use this symbol to indicate inversion. The symbol "%" gives the remainder of the division operation performed by "/".

Relational Operators

Relational operators perform various comparisons of elements in an expression and yield a Boolean true or false based on the result of the comparison. These operators greatly simplify the description of magnitude comparisons and reduce an identity comparison to a single statement. All relational operations are unsigned; care must be taken when negative numbers are represented in twos compliment. *Table 3* lists the relational operators.

Table 3. ABEL Relational Operators

Operator	Definition	Example
= =	equal	C=(A= =B);
!=	not equal	C=(A!=B);
<	less than	C=(A<B);
>	greater than	C=(A>B);
< =	less than or equal	C=(A< =B);
> =	greater than or equal	C=(A> =B);

Relational operators are frequently used where ranges of values cause a given output. For example, if a certain active low chip select line (CS1) is to be decoded for any address from ^h2000 and ^h2FFF, the logic for this output could be written in a single line as:

```
!CS1 = (ADD > = ^h2000) & (ADD< = ^h2FFF);
```

Assignment Operators: Combinatorial and Registered

Note that all example operations shown so far are for purely combinatorial outputs. The structure for combinatorial equations is:

```
OUTPUT(s) = Expression(s) and/or Condition(s);
```

The assignment operator is the "=", meaning that OUTPUT(s) will combinatorially follow the evaluation of the expressions and conditions. If an output or set of outputs is registered (changing synchronously with the rising edge of the clock), the assignment operator ":=" is used. The structure of a registered equation, shown below, is essentially the same as a combinatorial equation with the exception of this assignment operator:

```
OUTPUT(s) := Expression(s) and/or Condition(s);
```

Operator Priority

Operators in an expression are evaluated using a hierarchy of priority. If two or more operators with equal priority are used in a single expression, they are evaluated in the order listed from left to right within the expression. *Table 4* lists the priority of all operators.

Table 4. ABEL Operator Priority.

Highest Priority
- Twos compliment, not subtraction
! Inversion, ones compliment
Second Highest Priority
< Shift left
> Shift right
* Multiply
/ Unsigned division
% Remainder from division
Third Highest Priority
+ Add
- Subtract
OR
$ XOR
!$ XNOR
Lowest Priority
All Relational Operators
(= =, !=,<,>,< =,> =)

Parentheses may be used as in normal mathematics to alter the order of evaluation. The operation in the innermost parentheses is performed first.

Special Constants

Several special constants are supported that ease the writing of equations and test vectors. *Table 5* lists these special constants and their functions.

Table 5. ABEL Special Constants

Special Constant	Definition
.C.	Clock: causes a low-high-low transition at a selected input for testing.
.F.	Floating input or output
.K.	Same as .C., but high-low-high
.P.	Register preload
.X.	Don't care condition
.Z.	Tests input or output for high impedance

In order to use several of these constants in an abbreviated form and to enable the symbols "H" and "L" to represent binary ones and zeros, the following statement is placed in the labels section of all source documents included in this brief:

```
H,L,X,C,Z  =  1,0,.X.,.C.,.Z.;
```

Logic Reduction Levels

At the beginning of every source file in this brief, the statement

```
flag '-r4'
```

is used. This signals the language processor to use logic reduction level 4. In cases where propagation delays of a specific length are required, the statement

```
flag '-r0'
```

is used, which indicates no reduction may be used. Four levels of logic reduction are available to the designer as listed in *Table 6.*

Table 6. ABEL Logic Reduction Levels

Level	Statement	Description
0	flag '-r0'	No reduction. All equations must be in sum-of-products form.
1	flag '-r1'	Equations are expanded to sum-of-products form and reduced with standard Boolean algebra. This is the default.
2	flag '-r2'	Includes level 1 reduction plus the PRESTO algorithm. This process is iterative, so processing time is increased significantly.
3	flag '-r3'	The PRESTO algorithm is performed on a pin-by-pin basis. This is faster than standard PRESTO reduction.
4	flag '-r4'	This reduction level uses the ESPRESSO reduction algorithm.

ABEL Design Entry: Boolean Equations

This is the most common method of design entry. Each pin required for a given application is given a name. If a design requires the use of the special functions (i.e., reset and preset) that are available in many devices, the nodes that control these functions are also identified and named. The PALC22V10 has two such nodes; Asynchronous Reset (node 25) and Synchronous Preset (node 26). Groups of pins and/or frequently used constants may then be given labels to facilitate writing equations.

Following the keyword "EQUATIONS" in the source file, Boolean equations using the pin, node, and/or label names are generated to describe the required logic.

If an output has an output enable term associated with it, the user may write an equation for that term by using the pin name with the extension ".OE" followed by the equation for the term. An example of this is:

```
OUT1.OE = !RD & (INPUTS = = 0);
```

This statement causes OUT1 to be enabled if pin RD is low and the group of pins (can be any number of pins) labeled INPUTS are all low. If these conditions are not met, the output remains tri-stated.

The PALC22V10 has a separate combinatorial output enable product term for each I/O pin. The output enable is therefore easily controlled by either a single, selectable pin or from a product term. To make an output enable synchronous or to expand the number of product terms available, an I/O macrocell can be dedicated to realizing the appropriate logic with the output of the macrocell being fed back to control the OE product term. However, this method causes additional propagation delay due to the extra pass through the AND/OR array.

The use of the enable equations is purely optional; in the absence of any such equations, the ABEL language processor automatically enables any I/O pin that is defined in the Boolean equations as an output (appears on the left side of an equation) and disables any I/O that is specified as an input.

The operators and syntax of all Boolean equations are outlined in this brief. Additional information can be found in the ABEL Language Reference and User's Guide that are supplied with the ABEL software from DATAI/O®

ABEL Design Entry: Truth Tables

A truth table is a list of input combinations and the resulting outputs. Normally, the inputs will be listed in ascending binary order from the minimum value to the maximum value. This takes all possible input situations into account and prevents any undefined input combinations from producing undesirable outputs.

The keyword "TRUTH_TABLE" marks the beginning of the table within the source file. Immediately following the keyword, the input(s) and output(s) labels are listed in parentheses with an arrow (composed of a minus sign and a greater than sign "->") between the inputs and outputs. If more than one input or output is specified, square brackets "[ ]" must enclose the set. *Figure 1* illustrates the statements required to implement a 3 to 8 line decoder. Note the use of the set identifier Q7..Q0. This could have been written out as Q7,Q6,Q5,Q4,Q3,Q2,Q1,Q0.

```
truth_table
([I2,I1,I0] -> [Q7..Q0])
 [0,0,0] -> [0,0,0,0,0,0,0,1];
 [0,0,1] -> [0,0,0,0,0,0,1,0];
 [0,1,0] -> [0,0,0,0,0,1,0,0];
 [0,1,1] -> [0,0,0,0,1,0,0,0];
 [1,0,0] -> [0,0,0,1,0,0,0,0];
 [1,0,1] -> [0,0,1,0,0,0,0,0];
 [1,1,0] -> [0,1,0,0,0,0,0,0];
 [1,1,1] -> [1,0,0,0,0,0,0,0];
```

Figure 1. Truth Table for 3:8 Line Decoder

The main advantage of the truth table entry method is found in writing test vectors; the entire truth table can be block copied to the test vector section of the source file.

Any design specified by a truth table can be alternately entered as boolean equations. For example, the output Q6 in the above example could be represented by the Boolean equation:

```
Q6 = I2 & I1 & !I0;
```

ABEL Design Entry: State Machine Syntax

One of the most powerful features of the ABEL programming language is its ability to directly compile state diagrams. By allowing direct state diagram entry, ABEL frees the designer from the tedious task of generating Boolean equations that include the expressions and conditions that cause each possible transition for each individual state register.

The state machine syntax for each set of outputs (several state machines can be implemented in a single device) begins with the keyword "state_diagram" followed by the pin names or labels that make up the state outputs. Each state is then listed followed by any operations to be performed while in that state and at least one transition statement. A transition statement can be in any of three forms:

1. GOTO - for unconditional transitions to the next state.
2. IF..THEN..ELSE - for two-way branching.
3. CASE..ENDCASE - for N-way branching.

IF..THEN..ELSE statements may be chained to achieve n-way branching, but the CASE..ENDCASE construct accomplishes the same thing with less typing. Use of labels for state outputs and condition inputs enables even the most complex designs to be implemented with ease. As an example, consider a bi-directional 3-bit counter with inputs UP and DOWN and outputs Q2,Q1, and Q0. If UP or DOWN is high the counter is to count in the direction specified. If both UP and DOWN are high, the counter should hold the current count. If both UP and DOWN are low, the counter should reset to zero. In addition, let output MAX be high if the counter is in the UP mode and the count equals 7 or if the counter is in the DOWN mode and the count equals zero. Convenient labels for implementing such a design are shown in *Figure 2.*

```
"labels
OUTS = [Q2..Q0];
MODE = [UP,DOWN];
CNTUP = ^b10; CNTDWN = ^b01;
RST  = ^b00; HOLD  = ^b11;
S0 = ^b000; S1 = ^b001; S2 = ^b010;
S3 = ^b011; S4 = ^b100; S5 = ^b101;
S6 = ^b110; S7 = ^b111;
```

Figure 2. State Machine Labels for Counter Example

For the required operation, the state diagram for this design is listed in *Figure 3.*

```
state_diagram OUT
state S0: MAX = (MODE = = CNTDWN);
        case (MODE = = CNTUP) : S1;
             (MODE = = CNTDWN) : S7;
             (MODE = =HOLD) : S0;
             (MODE = = RST)    : S0;
        endcase;
state S1 : MAX = 0;
        case (MODE = = CNTUP)  : S2;
             (MODE = = CNTDWN) :S0;
             (MODE = = HOLD)  : S1;
             (MODE = = RST)    : S0;
        endcase;
state S2 : MAX = 0;
        case  (MODE = = CNTUP)  : S3;
              (MODE = =CNTDWN):S1;
              (MODE = = HOLD)  : S2;
              (MODE = = RST)  : S0;
        endcase;
state S3 : MAX = 0;
        case (MODE = = CNTUP) : S4;
             (MODE = =CNTDWN) :S2;
             (MODE = = HOLD)  : S3;
             (MODE = = RST)  : S0;
        endcase;
state S4 : MAX = 0;
        case (MODE = = CNTUP) : S5;
             (MODE = = CNTDWN): S3;
             (MODE = = HOLD)  : S4;
             (MODE = = RST)    : S0;
        endcase;
state S5 : MAX = 0;
        case (MODE = = CNTUP)  : S6;
             (MODE = = CNTDWN) : S4;
             (MODE = = HOLD)  : S5;
             (MODE = = RST)  : S0;
        endcase;
state S6 : MAX = 0;
        case (MODE = = CNTUP) : S7;
             (MODE = = CNTDWN) : S5;
             (MODE = = HOLD)  : S6;
             (MODE = = RST)    : S0;
        endcase;
state S7 :MAX = (MODE = = CNTDWN);
        case (MODE = = CNTUP) : S0;
             (MODE = = CNTDWN): S6;
             (MODE = = HOLD)  : S7;
             (MODE = = RST)    : S0;
         endcase;
```

Figure 3. ABEL Source Code for Counter Example

An additional statement, WITH..ENDWITH, can be added to any transition statement. This allows additional outputs to be set to any given state when the transition preceding the WITH..ENDWITH statement is executed. For example, in the previous state diagram, assume a pin called FLAG is to be set by the transition from state S5 to S6. The S5 diagram would be modified as shown in *Figure 4.*

```
state S5 : MAX = 0;
        case (MODE = = CNTUP)  : S6
                with FLAG := 1;
                endwith
             (MODE = = CNTDWN): S0;
             (MODE = = HOLD)  : S5;
             (MODE = = RST)    : S0;
        endcase;
```

Figure 4. WITH..ENDWITH Example

PALC22V10 Design Examples

The following design examples exploit the various features of the PALC22V10 Programmable Logic Device. The first seven designs focus on particular features and illustrate the techniques for using and testing these features. The last three designs combine several of the features to demonstrate the device's versatility. It is the tremendous versatility of this device that has made it the most popular of all Cypress PLDs. Each of the last three designs, if implemented in SSI and MSI TTL logic, would require from seven to thirteen packages.

Asynchronous Reset/Synchronous Preset, from a Single Pin

This example, as shown in *Figure 5*, defines pins 2 and 3 to be the Asynchronous Reset and Synchronous Preset inputs, respectively. Eight inputs defined as INPUT7..INPUT0 are given the label INPUTS. Eight corresponding outputs, OUTPUT7..OUTPUT0, are labeled OUTPUTS. Note how the use of labels enables the logic for all eight outputs to be written in a single equation. The equation:

```
OUTPUTS := INPUTS;
```

causes the data at INPUTS to be registered in OUTPUTS on the rising edge of CLK. The operation is indicated to be clocked (registered) by the assignment operator ":=". The clock input on the PALC22V10 is, by definition, pin 1.

```
                                                          "Cypress Semiconductor Corp. 11/10/1987

module Rst_Pre1                                           "Module name test

      flag '-r3'                                          "Logic Reduction level r3,  fast PRESTO
      title 'Asynchronous Reset / Synchronous Preset Control From A Single Input

                                                          "Device designator and type
      U1 device 'P22V10';
                                                          "Pin assignments
      CLK                                   pin 1;        "Clock input
      RST                                   pin 2;        "Defines async reset pin
      PRE                                   pin 3;        "Defines sync preset pin
      INPUT7,INPUT6,INPUT5,INPUT4           pin 4,5,6,7;
      INPUT3,INPUT2,INPUT1,INPUT0           pin 8,9,10,11;
      OUTPUT7,OUTPUT6,OUTPUT5,OUTPUT4       pin 23,22,21,20;
      OUTPUT3,OUTPUT2,OUTPUT1,OUTPUT0       pin 19,18,17,16;
      reset,preset                          node 25,26;   "Pre-assigned node #s

                                                          "Labels
      H,L,X,C,Z       =    1,0,.X.,.C.,.Z.;
      INPUTS          =    [INPUT7..INPUT0];
      OUTPUTS         =    [OUTPUT7..OUTPUT0];

      @radix 16;                                          "This command forces the default
                                                          "number base to HEX.
      equations
      reset           =    !RST;                          "Async reset when pin RST low
      preset          =    PRE;        "Sync preset if pin PRE is high during the rising edge of CLK
      OUTPUTS         :=   INPUTS;     "The := indicates that this a clocked (synchronous) operation
      test_vectors
                                                     "Test reset and preset
      ([CLK,RST,PRE,INPUTS] ->   OUTPUTS)
      [C,H,L,55]      ->   55;                       "Test outputs by clocking in 55
      [L,H,L,0AA]     ->   55;                       "Test registers hold old data (55)
      [C,H,L,0AA]     ->   0AA;                      "Clock AA (leading zero necessary for hex digits A-F)
      [C,H,L,0FF]     ->   0FF;                      "Set all outputs high (FF)
      [L,L,L,0FF]     ->   0;                        "RST low asynchronously
      [C,H,H,0]       ->   0FF;                      "PRE high synchronously
end Rst_Pre1
```

Figure 5. Reset/Preset from Single Pins

The predefined node numbers for the reset and preset functions are identified in the pin assignments section. The equations for the nodes in terms of the selected pins are then written in the equations section of the file.

```
                                          "Cypress Semiconductor Corporation, 11/10/1987
module Rst_Pre2                           "Module name test
flag '-r3'                                "Logic Reduction level r3, PRESTO algorithm by pin
title 'Asynchronous Reset / Synchronous Preset Example 2, Reset and Preset generated from Product terms'
                                    "*************************************************************
                                    "* This Example will Asynchronously Reset all registers when the inputs
                                    "* Synchronously Set all registers when the inputs equal AA
                                    "*************************************************************
                                    "Device designator and type
U1 device 'P22V10';
                                                    "Pin assignments
CLK                                 pin 1;          "Clock input
INPUT7,INPUT6,INPUT5,INPUT4         pin 4,5,6,7;
INPUT3,INPUT2,INPUT1,INPUT0         pin 8,9,10,11;
OUTPUT7,OUTPUT6,OUTPUT5,OUTPUT4     pin 23,22,21,20;
OUTPUT3,OUTPUT2,OUTPUT1,OUTPUT0     pin 19,18,17,16;
reset,preset                        node 25,26;     "Pre-assigned node #s

                                                    "Labels
H,L,X,C,Z       =       1,0,.X.,.C.,.Z.;
INPUTS                  =       [INPUT7..INPUT0];
OUTPUTS         =       [OUTPUT7..OUTPUT0];

@radix 16 ;command forces the default number base to be HEX

equations
reset           =       (INPUTS = =55);     "Async reset when input = 55
preset          =       (INPUTS = =0AA);    "Sync preset if inputs = AA during the rising edge of CLK
OUTPUTS:        :=      INPUTS;             "The := indicates that this a clocked (synchronous) operation

test_vectors                        "Test reset and preset
([CLK,INPUTS] -> OUTPUTS)
[C,0]    ->     0;                  "Test outputs by clocking in 0
[L,0FF] ->      0;                  "Test registers hold old data (0)
[C,0FF] ->      0FF;                "Clock in FF (note leading zero for hex digits A thru F)
[L,55]   ->     0;                  "RST low asynchronously on inputs = 55
[L,0AA]->       0;                  "No change, PRE is synchronous
[C,0AA]->       0FF;                "PRE acts synchronously on inputs = AA
end Rst_Pre2
```

Figure 6 . Reset / Preset From Product Terms

Asynchronous Reset/Synchronous Preset from Product Terms

This example, as shown in *Figure 6,* is similar to the example in *Figure 5* except that the reset and preset nodes are now activated from product terms. In particular, the reset node is high (active) only when INPUTS equal 55 hex. The preset term is similarly controlled by INPUTS equaling AA hex. Note how the test vectors distinguish and test the synchronous versus the asynchronous operations.

Using Reset and Preset to Load Predetermined Values

In the examples in *Figures 5* and *6*, the positive, registered output of the macrocells for the pins represented by OUTPUTS were used. This causes asynchronous reset to cause all outputs to go low and synchronous preset to cause them to go high.

This example demonstrates how "istype" statements, included in the pin assignments section, can be used to set any pattern of ones and zeros either asynchronously with reset or synchronously with preset. Four paths exist from the macrocells to the I/O pins. The Q and NOT Q outputs of each macrocell's register and the true and inverted combinatorial terms that bypass the register pass through a 4:1 multiplexer. The multiplexer is controlled by architecture bits C0 and C1, pictured in the macrocell diagram in *Figure 7* .

The istype statements allow the designer to select which channel of the multiplexer is routed to the I/O pin. The choices available are shown in *Table 8.*

Table 8. Macrocell Configuration Selections

C1	C0	Configuration	istype Values
0	0	Reg,Active Low	'neg, reg'
0	1	Reg,Active High	'pos, reg'
1	0	Comb,Active Low	'neg, com'
1	1	Comb,Active High	'pos, com'

An additional parameter in the istype statement allows for selection of feedback paths. The choices are feed_term, feed_reg, and feed_pin. An example showing this parameter is:

OUTPUT6 istype 'pos,com,feed_pin';

The PALC22V10 does not offer a feedback path from product terms and the selection of a feedback path is controlled by the same architecture bit (C1) that controls the selection of registered or combinatorial outputs. To specify a feedback path for this device would therefore be redundant.

Note from the test vectors in *Figure 8* that the use of istype statements does not affect the polarity of the outputs as described by the Boolean equations. Conversely, if an output is defined as active low through a boolean equation as in:

!OUTPUT6 := INPUT6;

the state of the register is inverted for both normal operation and for reset and preset conditions.

A final note on using istype statements in conjunction with the reset node: the PALC22V10 resets when Vcc is first applied to the chip. Istype statements and active low Boolean equations give the designer the opportunity to force the device's outputs to any desired state upon power up.

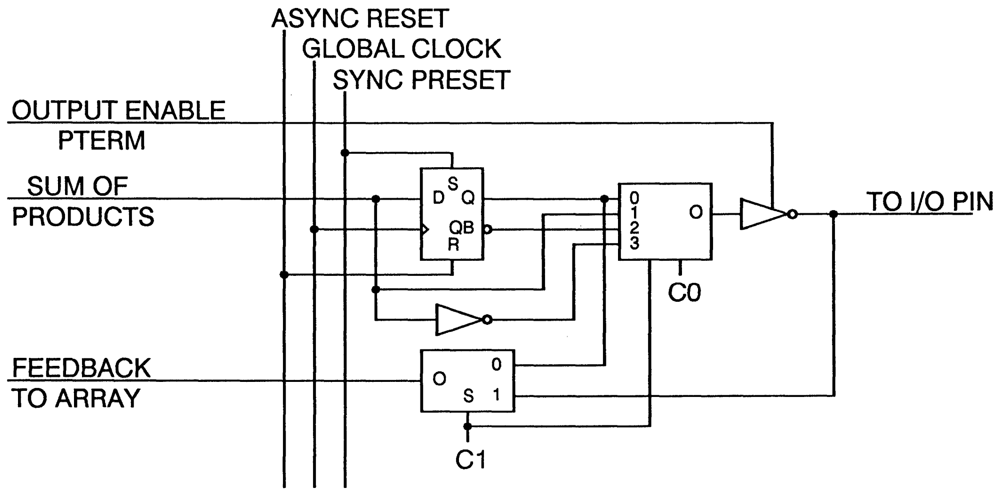


Figure 7. The PALC22V10 Macrocell

```
                                                "Cypress Semiconductor Corporation, 11/10/1987
module Rst_Pre3                                 "Module name test
       flag '-r3'                               "Logic Reduction level r3, PRESTO algorithm by pin

       title 'Asynchronous Reset/Synchronous Preset Example 3, Using Reset and Preset to Load to Predetermined States
       "*************************************************************************
       "* This Example will Asynchronously Load a Value of 55 and Synchronously Load     *
       "* Value of AA by using 'istype' statements to invert alternating output registers  *
       "*************************************************************************
                                         "Device designator and type
       U1 device 'P22V10';
                                                            "Pin assignments
       CLK                                      pin 1;      "Clock input
       RST                                      pin 2;      "Defines async reset pin
       PRE                                      pin 3;      "Defines sync preset pin
       INPUT7,INPUT6,INPUT5,INPUT4              pin 4,5,6,7;
       INPUT3,INPUT2,INPUT1,INPUT0              pin 8,9,10,11;
       OUTPUT7,OUTPUT6,OUTPUT5,OUTPUT4          pin 23,22,21,20;
       OUTPUT3,OUTPUT2,OUTPUT1,OUTPUT0          pin 19,18,17,16;
       OUTPUT7,OUTPUT5,OUTPUT3,OUTPUT1          istype 'pos,reg'; "Odd regs positive logic
       OUTPUT6,OUTPUT4,OUTPUT2,OUTPUT0          istype 'neg,reg'; "Even regs negative
       reset,preset                             node 25,26; "Pre-assigned node #s
                                                "Labels
       H,L,X,C,Z      =    1,0,.X.,.C.,.Z.;
       INPUTS         =    [INPUT7..INPUT0];
       OUTPUTS        =    [OUTPUT7..OUTPUT0];

       @radix 16;                        "command forces the default number base to be HEX

       equations
       reset          =    !RST;         "Async reset when pin RST low
       preset         =    PRE;          "Sync preset if pin PRE is high during the rising edge of CLK
       OUTPUTS        :=   INPUTS;       "The := indicatese that this a clocked (synchronous) operation
       test_vectors
       ([CLK,RST,PRE,INPUTS] -> OUTPUTS) "Test Reset and Preset
       [C,H,L,55]     ->   55;           "Test outputs by clocking in 55
       [L,H,L,0AA]    ->   55;           "Test registers hold old data (55)
       [C,H,L,0AA]    ->   0AA;          "Clock in AA (note the leading zero necessary for hex digits A thru F)
       [C,H,L,0FF]    ->   0FF;          "Set all outputs high (FF)
       [L,L,L,0FF]    ->   55;           "RST low asynchronously (bits 6,4,2,0 inverted)
       [C,H,H,0]           ->    0AA;          "PRE high synchronously (bits 6,4,2,0 inverted)

end Rst_Pre3
```

Figure 8. Resetting and Presetting to Predetermined Values

Output Enable Controlled by a Single Pin

The example in *Figure 9* defines pin 2 to be the output enable pin for all outputs. Note the use of special constant ".Z." which is redefined as simply "Z" in the labels section of the file. The constant is used in the test vectors to verify the outputs are tristated (high-Z) under the appropriate conditions.

```
                                                   "Cypress Semiconductor Corporation November 10, 1987
module Out_Enable1                                 "Module name
       flag '-r3'                                  "Logic Reduction level r3
       title 'Output Enable from Single Input Example '
                                                   "**********************************************
                                                   "* This example demonstrates the Output Enable,    *
                                                   "* Function being controlled by a single input     *
                                                   "**********************************************
       U1 device 'P22V10';                         "Device designator and type
                                                   "Pin assignments
       CLK                                            pin 1;          "Clock input
       OE                                             pin 2;          "Output enable input
       INPUT7,INPUT6,INPUT5,INPUT4                    pin 4,5,6,7;
       INPUT3,INPUT2,INPUT1,INPUT0                    pin 8,9,10,11;
       OUTPUT7,OUTPUT6,OUTPUT5,OUTPUT4                pin 23,22,21,20;
       OUTPUT3,OUTPUT2,OUTPUT1,OUTPUT0                pin 19,18,17,16;
       reset,preset                                   node 25,26;     "Pre-assigned node #s
                                                   "Labels
       H,L,X,C,Z      =     1,0,.X.,.C.,.Z.;
       INPUTS         =     [INPUT7..INPUT0];
       OUTPUTS        =     [OUTPUT7..OUTPUT0];
       OUTEN          =     [OUTPUT7.OE..OUTPUT0.OE];

       @radix 16;                                  "This command forces the default number base to be HEX

       equations
       OUTEN          =     !OE;                   "Outputs enabled only if pin OE is low
       OUTPUTS        :=    INPUTS;

       test_vectors                                "Test output enables

       ([CLK,OE,INPUTS]     ->     OUTPUTS)
       [C,L,55]       ->    55;                    "Test outputs by clocking in 55 (outputs enabled)
       [L,H,0AA]      ->    Z;                     "Test outputs go to high-Z state on OE high
       [L,L,0AA]      ->    55;                    "Test registers hold old data (55)
       [C,L,0AA]      ->    0AA;                   "Clock in AA (note the leading zero necessary for hex digits A thru F)
       [C,H,0FF]      ->    Z;                     "Set all outputs high (FF) but tri-stated
       [L,L,X]        ->    0FF;                   "Turn outputs on and read FF
end Out_Enable1
```

Figure 9. Output Enable Controlled by a Single Input

```
                                        "Cypress Semiconductor Corp.  11/10/1987
module Out_Enable2                      "Module name
      flag '-r3'                        "Logic Reduction level r3
      title 'Output Enable From a Product Term Example'
                                        "**********************************************
                                        "* This example demonstrates the Output Enable    *
                                        "* Function being controlled by a product term    *
                                        "**********************************************

      U1 device 'P22V10';                     "Device designator and type
                                                           "Pin assignments
      CLK, OE                                 pin 1, 2;    "Clock and Output Enable inputs
      INPUT7,INPUT6,INPUT5,INPUT4             pin 4,5,6,7;
      INPUT3,INPUT2,INPUT1,INPUT0             pin 8,9,10,11;
      OUTPUT7,OUTPUT6,OUTPUT5,OUTPUT4         pin 23,22,21,20;
      OUTPUT3,OUTPUT2,OUTPUT1,OUTPUT0         pin 19,18,17,16;
      reset,preset                            node 25,26;  "Pre-assigned node #s

      H,L,X,C,Z     =    1,0,.X.,.C.,.Z.;                  "Labels
      INPUTS             =    [INPUT7..INPUT0];
      OUTPUTS       =    [OUTPUT7..OUTPUT0];

      @radix 16;                   "This command forces the default number base to be HEX

      equations                    "Each Output individually enabled if the corresponding digital code is applied at
                                   "inputs and OE is low
      OUTPUT0.OE = (INPUTS = = 0) & !OE;      OUTPUT1.OE = (INPUTS = = 1) & !OE;
      OUTPUT2.OE = (INPUTS = = 2) & !OE;      OUTPUT3.OE = (INPUTS = = 3) & !OE;
      OUTPUT4.OE = (INPUTS = = 4) & !OE;      OUTPUT5.OE = (INPUTS = = 5) & !OE;
      OUTPUT6.OE = (INPUTS = = 6) & !OE;
      OUTPUTS := INPUTS;
      test_vectors
      ([CLK,OE,INPUTS] -> [OUTPUT7..OUTPUT0])
      [C,H,55]       ->    [Z,Z,Z,Z,Z,Z,Z,Z];
      [L,H,0]        ->    [Z,Z,Z,Z,Z,Z,Z,Z];
      [L,L,0]        ->    [Z,Z,Z,Z,Z,Z,Z,1];
      [L,L,1]        ->    [Z,Z,Z,Z,Z,Z,0,Z];     "Loads 55, checks OE high overrides
      [L,L,2]        ->    [Z,Z,Z,Z,Z,1,Z,Z];     "all enable terms, then enables and
      [L,L,3]        ->    [Z,Z,Z,Z,0,Z,Z,Z];     "checks all outputs one at a time
      [L,L,4]        ->    [Z,Z,Z,1,Z,Z,Z,Z];
      [L,L,5]        ->    [Z,Z,0,Z,Z,Z,Z,Z];
      [L,L,6]        ->    [Z,1,Z,Z,Z,Z,Z,Z];
      [L,L,7]        ->    [0,Z,Z,Z,Z,Z,Z,Z];

end Out_Enable2
```

Figure 10. Separate Output Enables Controlled by Product Terms

Output Enables Controlled by Product Terms

While *Figure 9* illustrated gang control of all output enables via an input pin, *Figure 10* shows several outputs all with individual output enables generated from separate product terms.

As with reset and preset, output enables can be made synchronous or have the number of product terms extended by using a macrocell to generate the necessary logic and "looping back" the term via a feedback path. This method incurs additional propagation delay due to passing through the AND/OR array twice.

The special constant ".Z." is used in the test vectors for this design to verify the operation of outputs in the tri-stated (high-Z) mode.

An 8-Bit Identity Comparitor

This example (refer to the source code in *Figure 11*) points out how the variable product term architecture (16 product terms maximum) of the PALC22V10 enables direct implementation of logic that would require multiple feedback terms to implement in standard PLDs. (Standard 20 pin PLDs have only 8 product terms per output.)

An *n* bit comparitor requires 2 to the nth power product terms to implement. The 8 bit comparison is achieved here by decomposing the 8 bits into two 4 bit comparisons and using I/O pins 18 and 19 (these pins have 16 product terms each) for each 4 bit comparison. The results of each 4 bit comparison are available at these outputs one t_{pd} after a match is detected

Note how the inputs and outputs are used in more than one label (*Figure 11*). This facilitates writing equations and test vectors for the individual 4 bit fields and the complete 8 bit fields

Using Feedback to Realize More Than 16 Product Terms: A Single Output 9-Bit Identity Comparitor

This example is very similar to the example in *Figure 11*, except the DATA inputs are rearranged to enable the two 4 bit comparitor outputs to be fed back and ANDed with the result of the single, 9th bit compare. The result is a single DATA = INPUTS output called INEQDATA.

The disadvantage of this implementation is that an additional t_{pd} is incurred by feeding the individual 4 bit comparitor outputs back through the AND/OR array. Note that although the terms fed back to INEQDATA represent 34 (16 + 16 + 2) product terms, only three of the 8 product terms available at I/O pin 23 are used; each of the three individual compares have already been reduced to single signals by the time they reach the AND/OR array for this pin. The extra product terms could be used along with a separately defined input for cascading the design to n-bit length. This source code for this example is shown in *Figure 12.*

Bidirectional I/O: Bus Interface Data Trap with Answer-back

This example (refer to the source code in *Figure 13)* demonstrates the bidirectional I/O capabilities of the PALC22V10. An 8 bit pattern is supplied to INPUTS and is continuously compared to the data on DATA7..DATA0. This design was created for an application where DATA7..DATA0 was the data bus of a Z80 microprocessor. If the interrupt is enabled (pin INTRENBL is high), the 8 bit comparitor output drives pin INTR active (low). In response, the Z80 drives pin IDREQ high. This requests that the device that initiated the interrupt places its 8 bit ID code on the data bus. In this example, the ID code used is ^h55. Any code may be used by modifying the equation for DATA in the source file.

10-Bit Counter, Address Generator/Multiplexer

The application that inspired the example in *Figure 14* was the address generation circuitry for the front end of a high-speed data acquisition module. The design requires two modes of operation. In the ACQUIRE mode, the 10 address lines are generated by counters. In the READ mode, the same addresses are generated by a microprocessor's address lines. In the original design, quad 2:1 multiplexers were used to select which source, the counters or microprocessor, would actually provide the address information. The entire circuit, excluding the SRAM being addressed, consisted of 11 SSI and MSI TTL components. The example given here implements the equivalent circuitry in a single PALC22V10.

```
                                       "Cypress Semiconductor Corporation     November 10, 1987
module AllTerms                        "Module name
     flag '-r3'                        "Logic Reduction level r3, PRESTO algorithm by pin
     title 'Using 16 Product Terms; An 8-bit Identity Comparitor '
                            "**************************************************************************
                            "* In this design, an 8-bit word is presented at I/O pins 23,22,21,20,17,16,15 and 14.
                            "* These pins are used for inputs only in this example. The 8-bit word is compared, 4 bits
                            "* at a time, to inputs INPUT7..0. Combinatorial outputs COMPHI and COMPLO show
                            "* the result of each 4-bit comparison. Pins 19 and 18 are used as the comparitor outputs
                            "* since these pins have enough Product Terms (16) for the required 4-bit comparisons.
                            "**************************************************************************
     U1 device 'P22V10';               "Device designator and type
                                                         "Pin assignments
     CLK                                     pin 1;      "Clock input (NOT used)
     INPUT7,INPUT6,INPUT5,INPUT4             pin 4,5,6,7;
     INPUT3,INPUT2,INPUT1,INPUT0             pin 8,9,10,11;
     DATA7,DATA6,DATA5,DATA4                 pin 23,22,21,20;
     DATA3,DATA2,DATA1,DATA0                 pin 17,16,15,14;
     COMPHI,COMPLO                           pin 19,18;  "Comparator outputs
     reset,preset                            node 25,26; "Pre-assigned node #s
     H,L,X,C,Z      =    1,0,.X.,.C.,.Z.;
     INPUTSH        =    [INPUT7..INPUT4];               "High-order nibble
     DATAH          =    [DATA7..DATA4];
     INPUTSL        =    [INPUT3..INPUT0];               "Low-order nibble
     DATAL          =    [DATA3..DATA0];
     DATA           =    [DATA7..DATA0];                 "All 8 bits
     INPUTS              =    [INPUT7..INPUT0];

     @radix 16;

     equations
     COMPHI = (INPUTSH = = DATAH);                       "High-order nibble compare
     COMPLO = (INPUTSL = = DATAL);                       "Low-order nibble compare

     test_vectors
     ([DATA,INPUTS] - > [COMPHI,COMPLO])
     [0,0]     - >    [H,H];     [1,1]     - >   [H,H];    [2,2]     - > [H,H];
     [4,4]     - >    [H,H];     [8,8]     - >   [H,H];    [0F,0F]   - > [H,H];
     [0E,0E] - >      [H,H];     [0D,0D] - >     [H,H];    [0B,0B]   - > [H,H];
     [7,7]     - >    [H,H];     [0,0F]    - >   [H,L];    [0F0,0F] - > [L,L];
     [0F0,0]  - >     [L,H];     [0F0,0FF]- >    [H,L];

end AllTerms
```

Figure 11. Using 16 Product Terms : An 8-Bit Identity Comparitor

```
                                 "Cypress Semiconductor Corporation November 10, 1987

module CompFB                    "Module name
     flag '-r3'                      "Logic Reduction level r3, PRESTO algorithm by pin
     title 'Using Feedback to Realize more than 16 Product Terms; A Single Output, 9-bit Identity Comparitor '
               "*************************************************************************
               "* In this design, an 9-bit word is presented at pins 23,22,21,20,17,16,11,10 and 9.    *
               "* These pins are used for inputs only in this example. The 8 LSBs of the 9-bit word are *
               "* compared, 4 bits at a time, to inputs INPUT7..0. Combinatorial outputs COMPHI and  *
               "* COMPLO show the results of each 4-bit comparison. Pins 19 and 18 are used as the    *
               "* comparitor outputs since these pins have enough Product Terms (16) for the required *
               "* 4-bit comparison. The MSBs (bit 8) of DATA and are compared at output COMPMSB. *
               "* Outputs COMPMSB, COMPHI, and COMPLO are ANDED together to form output  *
               "* INEQDATA.                                                                 *
               "*************************************************************************
     U1 device 'P22V10';                                     "Device designator and type
                                                             "Pin assignments
     INPUT8,INPUT7,INPUT6,INPUT5,INPUT4    pin 1,2,3,4,5;
     INPUT3,INPUT2,INPUT1,INPUT0           pin 6,7,8,9;
     DATA8,DATA7,DATA6,DATA5,DATA4         pin 10,11,13,14,15;
     DATA3,DATA2,DATA1,DATA0               pin 16,17,20,21;
     COMPH,COMPL,COMPMSB,INEQDATA          pin 19,18,22,23; "Comparator outputs
     reset,preset                          node 25,26;      "Pre-assigned node #s
     H,L,X,C,Z      =     1,0,.X.,.C.,.Z.;
     INPUTSH        =     [INPUT7..INPUT4];                 "High-order nibble
     DATAH          =     [DATA7..DATA4];
     INPUTSL        =     [INPUT3..INPUT0];                 "Low-order nibble
     DATAL          =     [DATA3..DATA0];
     DATA           =     [DATA8..DATA0];                   "All nine bits
     INPUTS         =     [INPUT8..INPUT0];
     @radix 16;
     equations
     COMPH          =     (INPUTSH = = DATAH);              "High-order nibble compare
     COMPL          =     (INPUTSL = = DATAL);              "Low-order nibble compare
     COMPMSB        =     (INPUT8 = = DATA8);               "MSB compare
     INEQDATA       =     COMPH & COMPL & COMPMSB;          "Logical AND of all comparisons
     test_vectors
     ([DATA,INPUTS] - > [COMPH,COMPL,COMPMSB,INEQDATA])
     [0,0]     - >    [H,H,H,H];   [111,111]  - >   [H,H,H,H];
     [22,22]   - >    [H,H,H,H];   [44,44]    - >   [H,H,H,H];
     [88,88]   - >    [H,H,H,H];   [1FF,1FF] - >    [H,H,H,H];
     [0,100]   - >    [H,H,L,L];   [1FF,0FF] - >    [H,H,L,L];
     [1FE,1FF] - >    [H,L,H,L];   [1FE,1EE] - >    [L,H,H,L];
end CompFB
```

Figure 12. Realizing More Than 16 Product Terms Through Feedback: A 9-Bit, Single-Output Identity Comparitor

```
                                            "Cypress Semiconductor Corp., 11/10/1987
module BiDirect                             "Module name test
      flag '-r3'                            "Logic Reduction level r3, PRESTO algorithm by pin
      title 'Bi-Directional I/O A Bus Interface Data Trap with Answer-Back'
                     "****************************************************************************
                     "* This example compares the pattern at pins INPUTS to the data on data bus pins   *
                     "* DATA7..DATA0. Pin INTR is driven low if they match and INTRENBL (interrupt      *
                     "* enable) is high. Input IDREQ is then driven high, requesting ID code (^h55 in   *
                     "* this example) to be put on the data bus                                          *
                     "****************************************************************************
      U1 device 'P22V10';

      IDREQ, INTRENBL                       pin 2,3;          ", Output Enable, Interrupt Enable
      COMPL,INTR                            pin 19,18;        "Used in comparision of 4 LSBs
      INPUT7,INPUT6,INPUT5,INPUT4           pin 4,5,6,7;
      INPUT3,INPUT2,INPUT1,INPUT0           pin 8,9,10,11;
      DATA7,DATA6,DATA5,DATA4               pin 23,22,21,20;
      DATA3,DATA2,DATA1,DATA0               pin 17,16,15,14;
      reset,preset                          node 25,26;             "Pre-assigned node #s
      H,L,X,C,Z       = 1,0,.X.,.C.,.Z.;
      INPUTS = [INPUT7..INPUT0];                                    "All inputs
      INPUTH = [INPUT7..INPUT4];                                    "High order nibble of INPUTS
      INPUTL = [INPUT3..INPUT0];                                    "Low order nibble of INPUTS
      DATA = [DATA7..DATA0];                                        "All data I/Os
      DATAH = [DATA7..DATA4];                                       "High order nibble of DATA
      DATAL = [DATA3..DATA0];                                       "Low order nibble of DATA
      DATAOE = [DATA7.OE..DATA0.OE];
      IDCODE = ^h55;                                                "Identification code

      equations
      DATAOE=     IDREQ;                              "Enables ID output onto data bus
      DATA  =     IDCODE;                             "Identification code for device (^h55)
      COMPL=      (DATAL == INPUTL);                  "4 LSBs compare
      !INTR  =    (DATAH == INPUTH) & COMPL & INTRENBL; "INTR active low, All bits equal and
                                                            "interrupt enabled (INTRENBL high)
      test_vectors
      ([IDREQ,INTRENBL,DATA,INPUTS] -> [COMPL,INTR,DATA])
      [L,H,^h0F,^h1F]        ->     [H,H,X];          "Low nibble equal,high not equal
      [L,H,^h0F0,^h0F1]      ->     [L,H,X];          "High nibble equal, low not equal
      [L,L,^h0AA,^h0AA]      ->     [H,H,X];          "Test Interrupt Enable
      [L,H,^h0AA,^h0AA]      ->     [H,L,X];          "DATA = INPUTS, INTR goes active (low)
      [L,H,^h55,^h55]        ->     [H,L,X];
      [H,H,Z,X]              ->     [X,X,IDCODE];     "DATA pins output IDCODE (^h55)

end BiDirect
```

Figure 13. BiDirectional I/O : Bus Interface Data

```
module AddGenMux flag '-r3'                 "Cypress Semiconductor Corporation November 10, 1987
    title '10-bit Address Generation / Multiplexer IC'
                                            "*******************************************************
                                            "* This PLD design generates Address signals A0-A9.    *
                                            "* If Control signal MODE is high, the address signals *
                                            "* are the output of a 10-bit counter. If MODE is low  *
                                            "* the device passes uP Address lines UPADD0-UPADD9    *
                                            "*******************************************************
    AdrsGen device 'p22v10';
    CLK                                 pin 1;          "System Master Clock
    A0,A1,A2,A3,A4,A5,A6,A7,A8,A9       pin 14,15,16,17,18,19,23,22,21,20;
    UPADD0,UPADD1,UPADD2,UPADD3         pin 2,3,4,5;
    UPADD4,UPADD5,UPADD6,UPADD7         pin 6,7,8,9;
    UPADD8,UPADD9                       pin 10,11;
    MODE                                pin 13;
    reset,preset                        node 25,26;
    H,L,X,C,Z     =     1,0,.X.,.C.,.Z.;
    AOUT          =     [A9..A0];                   "Address Outputs
    UPADD               =     [UPADD9..UPADD0];     "uP Address Lines
    @radix 16;
    equations                                       "Boolean equations
    reset         =     (UPADD = = 0) & !MODE;      "Reset if uP Address = 00 and MODE is low
    AOUT          :=    ((AOUT + 1) & MODE)         "Count up if MODE high or
                        # (UPADD & !MODE);          "Pass UPADD if MODE low
    test_vectors                                    "Check Operation
    ([CLK,UPADD,MODE] -> AOUT)
    [X,0,L]  ->   0;                                "Checks Reset Function
    [C,X,H] ->    1;     [C,X,H]  ->  2;     [C,X,H] ->   3;     [C,X,H]->4;
    [C,X,H] ->    5;     [C,X,H]  ->  6;     [C,X,H] ->   7;     [C,X,H]->8;
    [C,X,H] ->    9;     [C,X,H]  ->  0A;    [C,X,H] ->   0B;    [C,X,H]->0C;
    [C,X,H] ->    0D;    [C,X,H]  ->  0E;    [C,X,H] ->   0F;    [C,X,H]->10;
    [C,111,L]->   111;   [C,222,L]->  222;   [C,44,L] ->  44;    [C,88,L]->88;
    [C,2EE,L]->   2EE;   [C,1DD,L] -> 1DD;   [C,3BB,L]->  3BB;   [C,377,L]->377;
    [C,155,L]->   155;   [C,2AA,L] -> 2AA;   [C,3FF,L]->  3FF;   [C,222,H]->00;
    [C,0FF,L]->   0FF;   [C,X,H] ->   100;
    [C,1FF,L]->   1FF;   [C,X,H] ->   200;              "Load to states where all 8 LSBs
    [C,2FF,L]->   2FF;   [C,X,H] ->   300;              "are high (uP mode), then toggle in
    [C,3FF,L]->   3FF;   [C,X,H] ->   0;                "counter mode
end AddGenMux
```

Figure 14. 10-Bit Address Generator/Multiplexer

Note the how the MODE pin in the equations for the AOUT outputs controls the source of the addresses. Also note the use of the asynchronous reset node; the reset term is generated by the condition of the MODE being set for microprocessor access (low) and the processor address itself being zero. Although the effect at the outputs (all outputs = zero) is the same as if the reset term was not included, it gives the processor a method of resetting all the registers to a known state before allowing the counters to free run again.

Timing Diagram for 10-bit Address Generator / Multiplexer

One of the more interesting features of the ABEL SIMULATE program is its ability to generate timing diagrams for specified pins based on the test vectors in a source file. Although the timing diagrams do not show propagation delays, they can be useful for verifying a device's in-circuit operation with a logic analyzer. The SIMULATE output file shown in *Figure 15* was generated with the command line:

```
simulate -iaddmux.out -oaddmux.sim -t4 -
        w1,2,3,4,5,13,14,15,16,17,18
```

The *-i* indicates the input file, which in this case is the intermediate output file created by ABEL's FUSEMAP program. The *-o* tells SIMULATE where (into which file) to write the results. The *-t4* specifies the trace level where waveforms are displayed and the *-w1..18* indicates which pins to show in the waveform output.

```
ABEL Version 2.00b  Data I/O Corp.
Address Generation / Multiplexer IC
Simulate device AdrsGen, type 'P22V10'

            U   U   U   u
            P   P   P   P
            A   A   A   A   M
        C   D   D   D   D   O
        L   D   D   D   D   D   A   A   A   A
        K   0   1   2   3   E   0   1   2   3

V 0001  |   |   |   |   |   |_  |_  |   |   |
V 0002  C   |   |   |   |     |  _| |_  |   |
V 0003  C   |   |   |   |     | |_    | |   |
V 0004  C   |   |   |   |     |  _|  _| |_  |
V 0005  C   |   |   |   |     | |_  |     | |
V 0006  C   |   |   |   |     |  _| |_    | |
V 0007  C   |   |   |   |     | |_    |   | |
V 0008  C   |   |   |   |     |  _|  _|  _| |_
V 0009  C   |   |   |   |     | |_  |   |     |
V 0010  C   |   |   |   |     |  _| |_  |     |
V 0011  C   |   |   |   |     | |_    | |     |
V 0012  C   |   |   |   |     |  _|  _| |_    |
V 0013  C   |   |   |   |     | |_  |     |   |
V 0014  C   |   |   |   |     |  _| |_    |   |
V 0015  C   |   |   |   |     | |_    |   |   |
V 0016  C   |   |   |   |     |  _|  _|  _|  _|
V 0017  C   |_  |   |   |    _| |_  |   |   |
V 0018  C    _| |_  |   |   |    _| |_  |   |
V 0019  C   |    _| |_  |   |   |    _| |_  |
V 0020  C   |   |    _| |_  |   |   |    _| |_
V 0021  C   |   |_  |_    | |   |   |_  |_    |
V 0022  C   |_   _|   |   | |   |_   _|   |   |
```

Figure 15. ABEL Simulated Waveform

More information on the use of SIMULATE can be found in the *ABEL User's Guide and Language Reference* that are supplied with the ABEL software from DataI/O.

Triple-State Machine

This final example demonstrates the power of the PALC22V10 when used as a synchronous state machine. The application was the redesign of the timing circuitry for a radar system. The system performs 12 DFTs on each set of quadrature data returned in three antenna beams that are gated for 9 ranges. The nonbinary nature of these numbers (3 beams, 9 ranges, and 12 speed bins) made generating the timing signals with counter circuits cumbersome.

This example creates three state machines in a single PALC22V10. As can be seen from the state diagrams, (shown in *Figure 16*) the filter state machine is free running. The beam state machine only changes states when the filter outputs are in their maximum condition. Similarly the gate information changes only if both the filter and beam outputs are at the maximum values.

Note the combined use of boolean equations and state diagrams. A separate state diagram is written for each state machine, but the transitions are dependant upon the condition of the other state outputs. Also of note is the extreme use of labels for pins, groups of pins, and the state outputs. This greatly simplifies the writing of the state machine syntax and test vectors.

When first compiled, the ABEL FUSEMAP routine indicated several outputs that had too many terms for the physical array of the corresponding I/O pin. By carefully arranging the I/Os, the design was made to fit. The flag '-r3' reduction statement made the fit possible without the tedium of generating and manually reducing Boolean equations from the state diagrams.

The test vectors for this design are of particular interest. Note how the @REPEAT command is used to cycle through 35 states in order to make the gate state outputs toggle. This powerful command lets 325 test vectors be described in a concise and manageable manner.

```
                                   "Cypress Semiconductor Corporation November 10, 1987
module Statexam flag '-r3'
    title 'Timing Generation TRIPLE State Machine for DFT Processor using a Cypress Semiconductor PAL C22V10'
        "************************************************************************
        "*  BEAM STATES - 0, 1, 2 (3 not used), GATE STATES - 0, 1, 2, 4, 5, 6, 8, 9, A
        "*  (3,7,B,C,D,E,F not used), FILTER STATES - 0, 1, 2, 4, 5, 6, 8, 9, A, C, D, E
        "*  (3,7,B,F not used)
        "************************************************************************
    U1 device 'P22V10';
    SYSCLK                  pin 1;
    START                   pin 2;                 "Used for reset/power-up
    AB0,AB1,AB2,AB3,AB4     pin 23,14,22,15,21;    "Pins are non-sequential to take advantage of
    AB5,AB6,AB7,AB8,AB9     pin 16,18,19,20,17;    "The variable number of product terms in the 22V10
    reset,preset            node 25,26;            "Pre-assigned node #s
    AB0,AB1,AB2,AB3,AB4     istype  'pos,reg';     "Unnecessary because ABEL will set architecture bits
    AB5,AB6,AB7,AB8,AB9     istype  'pos,reg';     "automatically - shown for example purposes only
    H,L,X,C,Z     =    1,0,.X.,.C.,.Z.;
    ABall         =    [AB9..AB0];
    FILT          =    [AB3..AB0];
    BEAM          =    [AB5,AB4];
    GATE          =    [AB9..AB6];
    @radix 16;
                            "Filter States - note missing states
    F0 = 00; F1 = 01; F2 = 02; F3 = 04; F4 = 05; F5 = 06; F6 = 08;
    F7 = 09; F8 = 0A; F9 = 0C; F10 = 0D; F11 = 0E;
                                   "Beam States
    B0 = 00; B1 = 01; B2 = 02;
                                   "Gate States
    G0 = 00; G1 = 01; G2 = 02; G3 = 04; G4 = 05; G5 = 06; G6 = 08; G7 = 09; G8 = 0A;
    equations
    reset = START;                 "Initialize to all lows on START
    state_diagram FILT
    State F0: GOTO F1; State F1: GOTO F2; State F2: GOTO F3; State F3: GOTO F4;
    State F4: GOTO F5; State F5: GOTO F6; State F6: GOTO F7; State F7: GOTO F8;
    State F8: GOTO F9; State F9: GOTO F10; State F10: GOTO F11; State F11: GOTO F0;
    state_diagram BEAM
    State B0: case  (FILT = = ^b1110)   : B1;
                    (FILT != ^b1110)    : B0;
        endcase;
    State B1: case  (FILT = = ^b1110)   : B2;    "Increment ONLY if
                    (FILT != ^b1110)    : B1;    "FILT is at max (0E)
        endcase;
    State B2: case  (FILT = = ^b1110)   : B0;
                    (FILT != ^b1110)    : B2;
        endcase;
```

Figure 16. Triple State Machine (part1)

```
    state_diagram GATE                          "Increments ONLY if BEAM and FILT are at max
    State G0: case    ((BEAM == ^b10) & (FILT == ^b1110))     : G1;
                      ((BEAM != ^b10) # (FILT != ^b1110))     : G0;
            endcase;
    State G1: case    ((BEAM == ^b10) & (FILT == ^b1110))     : G2;
                      ((BEAM != ^b10) # (FILT != ^b1110))     : G1;
            endcase;
    State G2: case    ((BEAM == ^b10) & (FILT == ^b1110))     : G3;
                      ((BEAM != ^b10) # (FILT != ^b1110))     : G2;
                      endcase;
    State G3: case    ((BEAM == ^b10) & (FILT == ^b1110))     : G4;
                      ((BEAM != ^b10) # (FILT != ^b1110))     : G3;
            endcase;
    State G4: case    ((BEAM == ^b10) & (FILT == ^b1110))     : G5;
                      ((BEAM != ^b10) # (FILT != ^b1110))     : G4;
            endcase;
    State G5: case    ((BEAM == ^b10) & (FILT == ^b1110))     : G6;
                      ((BEAM != ^b10) # (FILT != ^b1110))     : G5;
            endcase;
    State G6: case    ((BEAM == ^b10) & (FILT == ^b1110))     : G7;
                      ((BEAM != ^b10) # (FILT != ^b1110))     : G6;
            endcase;
    State G7: case    ((BEAM == ^b10) & (FILT == ^b1110))     : G8;
                      ((BEAM != ^b10) # (FILT != ^b1110))     : G7;
            endcase;
    State G8: case    ((BEAM == ^b10) & (FILT == ^b1110))     : G0;
                      ((BEAM != ^b10) # (FILT != ^b1110))     : G8;
            endcase;
  test_vectors              "Verifies devices operation
  ([SYSCLK,START] -> [GATE,BEAM,FILT])
  [X,H] -> [G0,B0,F0];   [C,L] -> [G0,B0,F1];   [C,L] -> [G0,B0,F2];[C,L] -> [G0,B0,F3];
  [C,L] -> [G0,B0,F4];   [C,L] -> [G0,B0,F5];   [C,L] -> [G0,B0,F6];[C,L] -> [G0,B0,F7];
  [C,L] -> [G0,B0,F8];   [C,L] -> [G0,B0,F9];   [C,L] -> [G0,B0,F10];[C,L] -> [G0,B0,F11];
  [C,L] -> [G0,B1,F0];   [C,L] -> [G0,B1,F1];   [C,L] -> [G0,B1,F2];[C,L] -> [G0,B1,F3];
  [C,L] -> [G0,B1,F4];   [C,L] -> [G0,B1,F5];   [C,L] -> [G0,B1,F6];[C,L] -> [G0,B1,F7];
  [C,L] -> [G0,B1,F8];   [C,L] -> [G0,B1,F9];   [C,L] -> [G0,B1,F10];[C,L] -> [G0,B1,F11];
  [C,L] -> [G0,B2,F0];   [C,L] -> [G0,B2,F1];   [C,L] -> [G0,B2,F2];[C,L] -> [G0,B2,F3];
  [C,L] -> [G0,B2,F4];   [C,L] -> [G0,B2,F5];   [C,L] -> [G0,B2,F6];[C,L] -> [G0,B2,F7];
  [C,L] -> [G0,B2,F8];   [C,L] -> [G0,B2,F9];   [C,L] -> [G0,B2,F10];[C,L] -> [G0,B2,F11];
  [C,L] -> [G1,B0,F0];          "Gate output changes state here
  @REPEAT ^D35 {[C,L] -> [X,X,X]; } [C,L] -> [G2,B0,F0];@REPEAT ^D35 {[C,L] -> [X,X,X]; } [C,L] -> [G3,B0,F0];
  @REPEAT ^D35 {[C,L] -> [X,X,X]; } [C,L] -> [G4,B0,F0];@REPEAT ^D35 {[C,L] -> [X,X,X]; } [C,L] -> [G5,B0,F0];
  @REPEAT ^D35 {[C,L] -> [X,X,X]; } [C,L] -> [G6,B0,F0];@REPEAT ^D35 {[C,L] -> [X,X,X]; } [C,L] -> [G7,B0,F0];
  @REPEAT ^D35 {[C,L] -> [X,X,X]; } [C,L] -> [G8,B0,F0];@REPEAT ^D35 {[C,L] -> [X,X,X]; } [C,L] -> [G8,B2,F11];
  [C,L] -> [G0,B0,F0];          "Check the final state rolls over to the first

                 "This completes a run-through of ALL states, the following 2 vectors retest reset (START)
  [C,L] -> [G0,B0,F1]; [C,H] -> [G0,B0,F0];
end Statexam
```

Figure 16. Triple State Machine (continued)

CYPRESS SEMICONDUCTOR

Using ABEL to Program the CY7C330

Introduction

ABEL is a very versatile logic design tool that has the capability of programing over three hundred different devices. Although the general documentation supplied with ABEL is reasonably good, device-specific information on the more complex PLDs is not always as thorough as a user would like.

The Cypress CY7C330 is a very powerful PLD. Features such as the input and buried registers allow the CY7C330 to fit into a wide variety of applications. The same features can make programming the device a challenge. This document describes how to access all the features of the Cypress CY7C330 using ABEL 3.0 or 3.1 with examples. The text contains references to the diagrams starting on page 4-82 of the 1989 Cypress Data Book.

For anyone still using ABEL 3.0 and trying to program the CY7C330 for the first time, there is a fatal flaw in the supplied device driver. Both Cypress and Data I/O have updated device drivers available. The device file supplied with ABEL 3.1 is correct.

When ABEL went from 3.0 to 3.1, they changed the name of the device file for the CY7C330. 'P330' was used for revision 3.0 and 'P330A' is used for 3.1, although 3.1 will still compile with the non-'A' device name. The only difference between these two device files is the syntax for specifying the shared input mux.

Input Registers

The CY7C330 contains 11 dedicated input macrocells. There is also an input register associated with each one of the 12 I/O macrocells which will be discussed later in the document.

Pins 3 and 14 have dual functionality. Pin 3, can be used as an input register or as clock input. Ten of the eleven input registers have the ability to be clocked from two different sources: pins 2 or 3. The choice of the clock source is individually programmable on a register-by-register basis. If an application requires only one input clock source, pin 3 can be used as a normal input. If an application requires both input clocks, pin 3 must be used as a clock input. There is a configuration bit that must be changed in order to enable pin 3 as a clock input.

The second of the dual functionality pins, pin 14, can be used as an registered input or as a global asynchronous output enable line. Control of the output enable on the 7C330 can originate from the product term array or from pin 14. The choice is programmable on a register-by-register basis. Control of the output enable will be explained in more detail in the I/O macrocell section.

There are two ways of controlling the input register clock mux. The most descriptive way is using the ".C" suffix as shown in the DEMO330.ABL example file supplied with ABEL 3.0/3.1. This method will work for the dedicated input registers (pins 4-7, 9-14) but will not work in ABEL 3.1 for the input registers in the I/O macrocells. The reason is that for the twelve I/O macrocells, ABEL thinks the clock mux is for the output or state register and not the input register.

The recommended method is using the macro commands. The macro file supplied with ABEL 3.0 does not include the complete list of macro needed to program all the clock muxes. The complete file is available from Cypress. This file, P330.INC, contains the macros needed to program all the clock muxes including the input registers. A listing of the macro file is located in *Appendix A*, at the end of this document. ABEL 3.1 comes with the complete macro file.

After the macro file is referenced in the ABEL source file, the pin 3 clock must be enabled by the command "CLK2". Then, setting particular clock muxes is done by entering "CLK2_n", where "n" is the pin number of the input register. This is shown in the example code below.

```
LIBRARY        'p330';
          "allows use of p330.inc macro file
CLK2;
          " enables pin3 as a clock input
CLK2_5 ;
          " pin 5 input  reg uses the pin 3 clock
CLK2_15;
          "pin 15 input reg  uses the pin 3 clock
```

No macro statement is needed to specify the use of clock 1 (pin 2) for input registers. Clock 1 is the clock mux default setting for both the dedicated input registers and the I/O macrocell input registers.

Accessing the data from one of the dedicated input registers (pins 3-14) is handled the same as a straight buffered input in ABEL . The only difference is that input data is not available in the product term array until after the appropriate input clock pulse is received.

Controlling the Output Enable

An output enable is specified by appending the suffix .OE to the appropriate pin name. The user must define whether control of the output enable mux comes from pin 14 or the product term array. This is controlled by configuration bit C0. The selection is made by using the ISTYPE statement as follows:

```
OUT1,OUT2,OUT3,OUT4   pin 15,16,17,18 ;
          " I/O pins

OUT1.OE,OUT2.OE ISTYPE 'EQN';
          " OE is product term controlled

OUT3.OE,OUT4.OE ISTYPE  'PIN';
          "OE is controlled by pin 14
```

When controlling the output enable with a product term, the user has the option of setting it always on, always off, or making it a combination of some number of inputs or outputs. All three choices are illustrated in the code below.

```
[OUT1.OE,OUT2.OE] = [1,1];
          "permanently enable outputs
OUT3.OE = 0;
          "permanently disable output
OUT4.OE = IN1 & IN2 & OUT1 ;
          "OE controlled by IN1, IN2, OUT1
```

Using Preset and Reset

The CY7C330 has global synchronous preset and reset capability. When used, it will set or reset all 12 state registers and the 4 buried macrocell registers. There are two things to watch out for when using set or reset. The first is, when you reset the registers, all the outputs go high if they are enabled. This is due to the inverter between the state register and the output as shown in the macrocell schematic in *Figure 1.* The second thing to watch out for is that the reset doesn't occur for two clock pulses if an input is designated as the set/reset pin. This is because the reset data must be clocked into the product term array using one of the two input clocks first. The output registers must then be clocked to cause the reset or set to occur.

There are two ways to access the set and reset capability of the CY7C330. The first is to append the suffix .PR for preset, or .RE for reset to any output pin or buried register node name. This syntax is shown below.

```
OUT1, INP1, INP2    PIN    16, 5, 6;

OUT1.PR  =  INP1 ;
          "preset all output nodes on INP1=1

OUT.RE  =  INP2;
          "reset all output nodes on INP2=1
```

The second way of utilizing set and reset is using the node notation shown below. The set and reset product terms have been given the designations node 30 and 29, respectively.

```
SET, RESET     NODE 30, 29 ;

SET = INP1 ;
          "preset all output nodes on INP1=1

RESET = INP2;
          "reset all output nodes on INP2=1
```

Even though the reset and preset functions are synchronous, an error will occur while parsing the equations if you use the ":=" notation, which is used to signify a registered operation.

Using the Macrocell as an Output Only

When using the I/O macrocell as an output, there are two parameters to be concerned with. The first is the setting of the macrocell feedback mux as controlled by configuration bit C1. The second parameter is the control of the output enable as described in the previous section. As with the output enable control, the configuration bit for the feedback mux is set using the ISTYPE statement. When the input register is not used data from the output register is typically fed back to the product term array through the macrocell feedback mux. In this case the ISTYPE will be followed by the 'FEED_REG' attribute as shown in the example below.

```
OUT1            PIN 15 ;
        "located in initial pin definitions

OUT1 ISTYPE 'FEED_REG';
        "sets C1=0, allowing feedback mux
        "to pass data from state register

OUT1 :=  INP1 $ ((INP1 & INP2 )# INP3);
        "sample eq from 'equations' section
```

The ABEL default for the feedback mux configuration bit (C1) is to take data from the state register. Thus the ISTYPE 'FEED_REG'; statement is not required, but it is recommended that the defaults be documented.

Using the Macrocell as an Input Only

When the I/O macrocell is used as an input register, the syntax is different. First, the output buffer must be tristated. Next, the macrocell feedback mux must be set to accept data from the input register (C1 must be set to 1). The following example assumes that the output register is not used at all. Keep in mind that the input register clock defaults to clock 1 (pin 2) unless specifically changed.

```
INP1, INP2, OUT2        PIN 5, 15, 16 ;

INP2 ISTYPE 'FEED_PIN';
        " set C1=0, allowing feedback mux to
        " take data from the  input register

INP2.OE  ISTYPE  'EQU';
        " set C0=0 for product term OE
EQUATIONS

INP2.OE  = 0 ;
        "tristate output buffer permanently
OUT2 :=  INP1 & INP2;
```

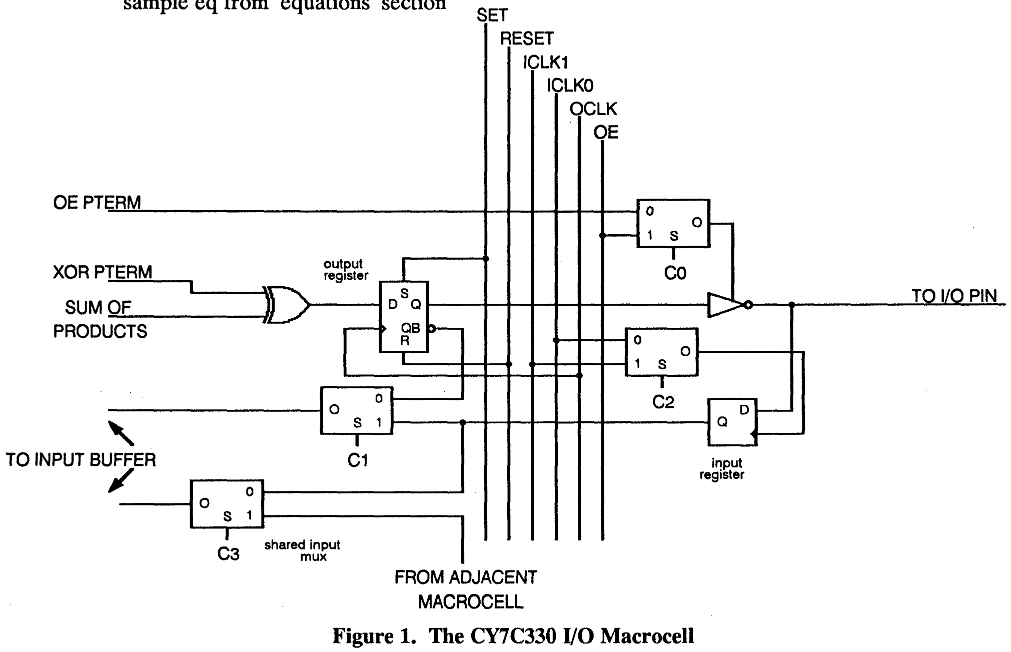


Figure 1. The CY7C330 I/O Macrocell

Shared Input Multiplexer

Each pair of I/O macrocells has a shared input mux. The shared input mux is used to feed data into the product term array if both registers are used in an I/O macrocell. A configuration bit (C3) controls whether the input of the mux will be from an even pin number macrocell or an odd. The ABEL default is that the data is supplied from the even pin number macrocell. Changing to an odd pin requires invoking macros located in the P330.INC file. The example in the next section shows how this is done.

The purpose of the shared input mux is to allow another path of feedback to the product term array. This means there are three feedback paths per pair of I/O macrocells. Thus, for every pair of macrocells only three out the four registers are available for use. If the resources of the CY7C330 are chosen wisely, this should not be much of a limitation.

Using the Input and Output Registers

When using both the input and output registers in the I/O macrocell, the most difficult task is getting the data into the product term array.

There are two muxes that can be used to feed data from the registers into the product term array. The state register information must be fed back through the feedback mux controlled by configuration bit C1. Input register data can be routed though the feedback mux or through the shared input mux. Refer to *Figure 1.*

The state register output is referred to by the pin name associated with the macrocell. The data being clocked into the input register is referred to by using the node name assigned to the shared input mux. The node numbers of the shared input muxes listed in *Table 1.*

Table 1. Shared Input Multiplexer Node Numbers

Node Number	Mux Between Pins
35	15, 16
36	17, 18
37	19, 20
38	23, 24
39	25, 26
40	27, 28

In ABEL, the configuration bit controlling the shared input mux (C3) defaults to an even I/O pin. When the input data is on an odd pin, a macro in the P330.INC macro file can be used to change the C3 configuration bit. The following example will also use clock 2 (pin 3) to clock the input register.

```
BREG          PIN 15 ;
        "BREG is output register for pin 15

INP1  NODE  35 ;
        "INP1 is the input register for pin 15

BREG  ISTYPE 'FEED_REG';
        "C1 is set to 0, mux routes Q of BREG
BREG  ISTYPE 'EQN';
        "OE is product term controlled

LIBRARY 'P330' ;
        "enables use of the P330.INC file
CLK2;
        "enables pin 3 clock
CLK2_15;
        " enables CLK2 on pin 15 input reg
FEEDPIN_15;
        "shared input mux control bit (C3) set
        " This gives pin 15 an input path

EQUATIONS

BREG.OE  =  0 ;
        "disable output

BREG  :=  BREG $ (INP1 & INP2);
        "BREG is fed back and INP1 is an input
```

The Exclusive OR Gate

On the D input of the 12 I/O macrocell output registers and the 4 buried macrocell registers is an exclusive OR (XOR) gate. This gate can be used for two purposes. The first is to invert the polarity of a signal going into the output register. This is accomplished by setting one of the XOR inputs to a logic 1. ("$" is the ABEL signal for XOR.) In the latest version of ABEL this can be done as shown in the following example:

```
OUT1 := 1 $ (INP1 & INP2 & INP3);
```

In earlier versions, however, the reduction algorithms will not recognize a "1" mixed with variables in an equation. The equivalent expression for earlier versions is:

```
OUT1 := (INP1 # !INP) $ (INP1&INP2&INP3);
```

The second use for the XOR gate is to do software emulation of JK or T flip-flops. T flip-flops are more

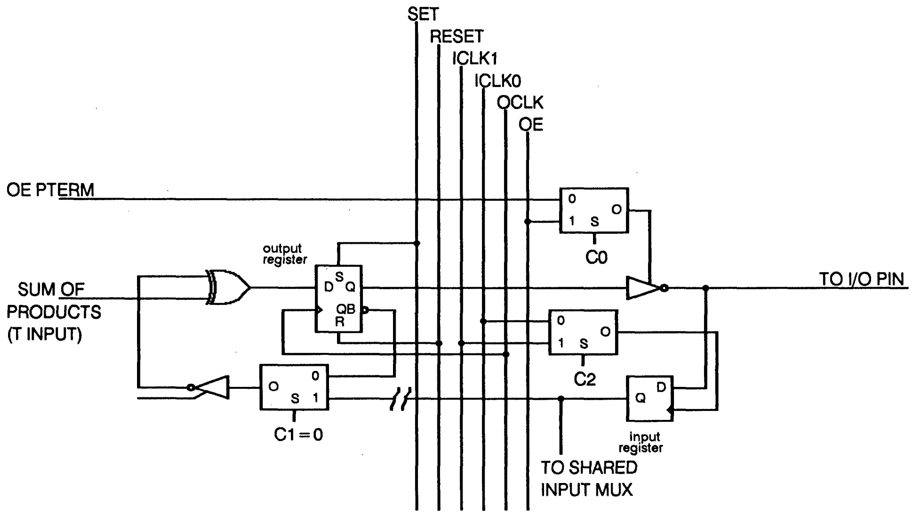


Figure 2. The CY7C330 Macrocell as a T-Type Flip-Flop

efficient than D flip-flops when building counters and state machines. Emulation of T-type flip-flops is accomplished by feeding back the output register's Q output and tying it to the XOR product term. The sum-of-products input to the XOR becomes the T input. Refer to *Figure 2.* This emulation can be done with boolean equations such as the one below:

TFLOP := TFLOP $ (T input expression);

where "T input expression" is a legal sum-of-products expression. A JK flip-flop is emulated using the same configuration, and the relationship:

T = J!Q # KQ

The second way to configure an output flip-flop as a T-type flop is to use an ISTYPE statement such as the one in the next example. Although this syntax works for simple state machines, ABEL is somewhat unpredictable in more complicated applications. The resulting reduced equations will not match the Jedec map, and the simulator may not work. The following syntax describes a simple 2 bit counter.

```
CLK, INSTB, !OE   PIN 1, 2, 3, 14;
Q0, Q1            PIN 28, 27;
Q0, Q1       ISTYPE        'REG_T' ;
Q0.OE, Q1.OE ISTYPE        'PIN';
CNT = [Q1,Q0];
EQUATIONS
Q0.OE = OE;
Q1.OE = OE;
CNT = (CNT + 1);
```

Buried Macrocells

As mentioned before, the CY7C330 contains 4 buried macrocells. Buried macrocells are accessed by assigning a name to the buried register node number. The node numbers are listed in *Table 2.*

Table 2. Node Numbers of Buried Registers

Buried Register	Node Number	Product Terms
1	31	13
2	32	17
3	33	11
4	34	19

A buried macrocell is pictured in *Figure 3.* To use a buried macrocell, assign a name to the node and use it as if it were a normal output. The only difference is that for the I/O macrocell there is an inverter between the state register and the output pin. The inverter causes ABEL to handle the polarity differently. This will be discussed in the next section.

Polarity Conventions

As shown in later examples, the designer typically does not have to worry about polarity of signals except when sending data to an output pin. The reason for this is that all data enters the product term array in both its

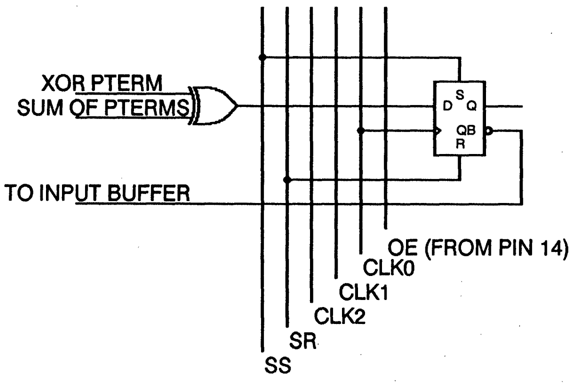


Figure 2. The CY7C330 Buried Macrocell

non-inverting and inverting state. ABEL will, in most cases, choose the right polarity to obtain the output as specified by the equations.

When data is being exported from the device via an output pin, polarity is more critical especially when using the set or reset. As shown by the I/O macrocell schematics, there is an inverter between the output register and output pin. Therefore, if you use the reset capability, the Q output of the registers goes low and the output pins go high. If your application requires all the outputs to start out low, use preset instead of reset.

In the following example, the output is defined as positive and a "1" and a "0" are passed through the device. ABEL compensates for the lack of inversion in the output by inverting the data coming out of the input register.

```
" inputs
CKS, CK1, CK2, INP  PIN  1, 2, 3, 4;
" output
OUT  PIN  15 ;

EQUATIONS

OUT := INP ;

TEST_VECTORS

([CKS,CK1,CK2,INP] -> [OUT])
[0, C, 0,  0] -> [X];
[C, 0, 0,  X] -> [0];
[C, 0, 0,  X]-> [0];
[0, C, 0,  1]-> [0];
[C, 0, 0,  X]-> [1];
[C, 0, 0,  X] -> [1];
END
```

When using state machine syntax, ABEL will not handle the polarity of the buried macrocells correctly. Not only will the equations not work, but the simulation will fail also. The problem can be easily fixed by negating the names in the node declaration as shown.

```
CLK1, CLK2, CLK3  PIN  1,2,3 ;
INP, OUT          PIN  4,15 ;
        "hidden register  declaration (negated)
!C1, !C2, !C3 NODE  31,32,33 ;
```

As with the state machine syntax, when using the 'COUNT = COUNT +1' syntax, you also must invert the polarity of any buried registers. The easiest place to accomplish the inversion is at the node definitions statement as shown in the previous example. Also, refer to the counter example at the end of this document.

State Machine Syntax

ABEL supports state machine syntax on the 7C330. The only drawback at this time is that the toggle flip-flop emulation mode can only be used with very simple state machines. As mentioned earlier, the results of using state machine syntax with T flip-flop emulation are unpredictable. The T flip-flop is much more efficient for state machines due to the fact that a toggle flip-flop only needs to use a product term for a state change. A toggle flip-flop will hold it's state unless told otherwise. A state machine using D flip-flops needs a product term both to change states and to hold states. Even with this limitation, the CY7C330 contains from nine to nineteen product terms per output and will usually handle a medium size state machine with ease. ABEL has promised that future releases will contain support for T flip-flops.

Simulation Caveats

There are limitations to what ABEL can and cannot simulate. The first limitation is that when writing simulation test vectors, only one of the three clock lines can be used on a single test vector line. The following example would not simulate correctly.

```
TEST_VECTORS

([CKS,CK1,CK2,INP] -> [OUT])
[C,C,0,0] -> [0];
```

This example will simulate correctly if modified as follows:

```
TEST_VECTORS
([CKS,CK1,CK2,INP]     -> [OUT])
 [0,C,0,0]             -> [X];
 [C,0,0,X]             -> [0];
```

The preload function is supported. Refer to the 15 bit counter example for more information on how to use it.

Example: 16-Bit Up/Down Counter

This first application, COUNTER6, is an example of a 15-bit up-counter with a terminal count output. The example shows how to use the 'COUNT = COUNT + 1' syntax of ABEL along with correcting the polarity problem that crops up when combining normal I/O macrocell output registers and buried macrocell registers. It also gives an example of using the preload function. The ABEL source code for this example can be found in *Appendix B.*

Example: Modulo 11 Counter Using State Machine Syntax

The second example is a basic state machine application implementing a - bit modulo-11 counter using state machine syntax. This again shows how to handle polarity using both normal registers and buried registers. The ABEL source code for this example can be found in *Appendix C.*

Appendix A. P330.INC - Macro Listing

```
" P330.INC
"The following select Clock 2 (pin 3) for the
"Output Macrocell Input register.
        CLK2_28 macro () {FUSES[17030] = 1;}
        CLK2_27 macro () {FUSES[17034] = 1;}
        CLK2_26 macro () {FUSES[17037] = 1;}
        CLK2_25 macro () {FUSES[17041] = 1;}
        CLK2_24 macro () {FUSES[17044] = 1;}
        CLK2_23 macro () {FUSES[17048] = 1;}
        CLK2_20 macro () {FUSES[17051] = 1;}
        CLK2_19 macro () {FUSES[17055] = 1;}
        CLK2_18 macro () {FUSES[17058] = 1;}
        CLK2_17 macro () {FUSES[17062] = 1;}
        CLK2_16 macro () {FUSES[17065] = 1;}
        CLK2_15 macro () {FUSES[17069] = 1;}

"The following enables clock 2 (pin 3)
        CLK2  macro () {FUSES[17070] = 1;}
        CLK2_4 macro () {FUSES[17072] = 1;}
        CLK2_5 macro () {FUSES[17073] = 1;}
        CLK2_6 macro () {FUSES[17074] = 1;}
        CLK2_7 macro () {FUSES[17075] = 1;}
        CLK2_9  macro () {FUSES[17076] = 1;}
        CLK2_10 macro () {FUSES[17077] = 1;}
        CLK2_11 macro () {FUSES[17078] = 1;}
        CLK2_12 macro () {FUSES[17079] = 1;}
        CLK2_13 macro () {FUSES[17080] = 1;}
        CLK2_14 macro () {FUSES[17081] = 1;}

"The following program the C3 bit in the Output Macrocell
"and selects feedback from the lower pin.
        FEEDPIN_27 macro () {FUSES[17031] = 1;}
        FEEDPIN_25 macro () {FUSES[17038] = 1;}
        FEEDPIN_23 macro () {FUSES[17045] = 1;}
        FEEDPIN_19 macro () {FUSES[17052] = 1;}
        FEEDPIN_17 macro () {FUSES[17059] = 1;}
        FEEDPIN_15 macro () {FUSES[17066] = 1;}
```

Appendix B. ABEL Source Code for 16-Bit Counter Example

```
module _counter6
title 'Counter application for 330 application note, Cypress Semiconductor June 19,1989'

counter6        device  'p330a' ;
                        " This is example of a 15 bit counter showing:
                        "1. How to handle the polarity when combining normal output registers and buried regs.
                        "2. How to use the 'count = count + 1' syntax.
                        "3. How to use preload for simulation vectors and handle the polarity inversion for the
                        "      buried registers.
" inputs pins
        clk,clk1,clk2,preset        pin     1,2,3,4 ;
" output pins
        c0,c1,c2,c3,c4,c5,c6 pin 15,28,26,17,24,19,20 ;
        c11,c12,c13,c14      pin  25,18,16,27 ;
        tci                  pin  23 ;
        spreset              node 30 ;
        !c7,!c8,!c9,!c10     node 31,32,33,34 ;
" macros
        c_cntr = [c14, c13, c12, c11, c10, c9, c8, c7, c6, c5, c4, c3, c2, c1, c0] ;
                        " this is used to handle the preload inversion of the buried registers. See test vectors below.
        c_cntrs = [c14, c13, c12, c11, !c10, !c9, !c8, !c7, c6, c5, c4, c3, c2, c1, c0] ;
        c,x,p          =        .c., .x., .p. ;
equations
        spreset        =        preset ;
        c_cntr :       =        (c_cntr + 1) ;
         tci           :=       (c_cntr = = 2346) ;

                        " Example of using preset with simulation

test_vectors
([clk,clk1,preset,c_cntrs] -> [c_cntr,tci])
 [0,0 , x , x    ]-> [ x    ,x];
 [0,c , 1 , x    ]-> [ x    ,x];
 [c,0 , x , x    ]-> [ 0    ,0];
 [0,c , 0 , x    ]-> [ 0    ,0];
 [c,0 , x , x    ]-> [ 1    ,0];
 [c,0 , x , x    ]-> [ 2    ,0];
 [c,0 , x , x    ]-> [ 3    ,0];
 [c,0 , x , x    ]-> [ 4    ,0];
 [c,0 , x , x    ]-> [ 5    ,0];
 [p,0 , x , 62  ]-> [ x    ,0];
 [0,0 , x , x    ]-> [ 62   ,0];
 [c,0 , x , x    ]-> [ 63   ,0];
 [c,0 , x , x    ]-> [ 64   ,0];
 [c,0 , x , x    ]-> [ 65   ,0];
 [c,0 , x , x    ]-> [ 66   ,0];
 [c,0 , x , x    ]-> [ 67   ,0];
 [c,0 , x , x    ]-> [ 68   ,0];
 [p,0 , x ,2345]-> [ x    ,0];
 [0,0 , x , x    ]-> [2345 ,0];
 [c,0 , x , x    ]-> [2346 ,0];
 [c,0 , x , x    ]-> [2347 ,1];
 [c,0 , x , x    ]-> [2348 ,0];
 [c,0 , x , x    ]-> [2349 ,0];
 end
```

Appendix C. ABEL State Machine Source Code for Mod11 Counter

```
module _statem
title 'Application Note State Machine Example,  Cypress Semiconductor  5-12-89'

        statem  device  'P330';

        clk1,clk2,clk3   pin     1,2,3 ;
        c1,c2            pin     15,16 ;
        res              pin 4 ;
        reset            node 30 ;
        !c3,!c4          node 31,32 ;
        count            =       [c4,c3,c2,c1] ;
        c4,c3,c2,c1              istype  'feed_reg' ;
        c,x,z,h,l        =       .c.,.x.,.z.,1,0 ;

                         " This is an example of implementing a modulo counter using state  machine syntax.
                         " This example also shows how to use the hidden registers.

                         " counter states
 s0 = ^b0000 ; s3 = ^b0011 ; s6 = ^b0110 ; s9 =   ^b1001;
 s1 = ^b0001 ; s4 = ^b0100 ; s7 = ^b0111 ; s10 = ^b1010 ;
 s2 = ^b0010 ; s5 = ^b0101 ; s8 = ^b1000 ;

equations
        c4.pr            =       res ;

state_diagram  [ c4,c3,c2,c1 ]
        state s0: goto s1 ;
        state s1: goto s2 ;
        state s2: goto s3 ;
        state s3: goto s4 ;
        state s4: goto s5 ;
        state s5: goto s6 ;
        state s6: goto s7 ;
        state s7: goto s8 ;
        state s8: goto s9 ;
        state s9: goto s10 ;
        state s10: goto s0 ;

test_vectors
([clk1,clk2,res] -> [count])
 [0 ,c ,1] -> [ 15 ];
 [c ,0 ,0] -> [ 0 ];
 [0 ,c ,0] -> [ 0 ];
 [c ,0 ,0] -> [ 1 ];
 [c ,0 ,0] -> [ 2 ];
 [c ,0 ,0] -> [ 3 ];
 [c ,0 ,0] -> [ 4 ];
 [c ,0 ,0] -> [ 5 ];
 [c ,0 ,0] -> [ 6 ];
 [c ,0 ,0] -> [ 7 ];
 [c ,0 ,0] -> [ 8 ];
 [c ,0 ,0] -> [ 9 ];
 [c ,0 ,0] -> [ 10 ];
 [c ,0 ,0] -> [ 0 ];

 end
```

NOTES:

CYPRESS SEMICONDUCTOR

CY7C330 66-MHz 28-Pin Synchronous EPLD

CY7C33X PLD Family

The Cypress CY7C330 is the first in a family of high-speed, application-optimized CMOS EPLDs. This fully synchronous part is designed for state machine and other clocked systems. The CY7C330 offers new solutions for systems designers, with a truly usable high-speed clock rate, 39 total registers, 17,000 programmable bits providing up to 1200 gate complexity. Other devices in the family are the CY7C331 and the CY7C332. All family members are packaged in 28-pin 300 mil dual inline and LCC/PLCC packages. The technology is low-power CMOS and UV-erasable.

The Cypress CY7C330 is the first application-specific EPLD from Cypress. The concept behind this family of high-speed devices, is to provide the optimal solution for each system design using Cypress's 0.8 micron, dual-level metal CMOS technology. Systems using other types of programmable logic devices for synchronous state machine applications, will use the CY7C330 as a higher density, lower power solution at speeds up to 66 MHz. The application-specific family from Cypress provides the CY7C330 for sequential state machine applications, the CY7C331 for general purpose asynchronous designs, and the CY7C332 for decoders and combinational logic applications.

The Cypress PALC22V10, PLDC20G10 and PAL20 devices proved the popularity of high-speed, low-power, erasable CMOS logic, and the CY7C330 builds on that base. One CY7C330 can easily replace four PALC22V10s by offering features such as extending the number of state registers to 16, extending the number of product terms per output to 19 maximum, and by adding the XOR logic function plus the ability to use pins as bidirectional I/O.

The CY7C330 design goal was to increase the speed of synchronous systems to 66 MHz. This is the actual usable speed, and is determined by the total 15 ns feedback time from the Q of a flip-flop to the D of any flip-flop in the device. The CY7C330 offers 258 variable product terms for 16 state registers. This allows very complex sequential machines to be designed with virtually no limitation of product terms. These designs can easily exceed the size anyone wants to manage with Karnaugh mapping. However, the new generation of advanced EPLD compilers can manage very complex state machine designs on workstations such as the IBM® PC/XT™.

In order to ensure the 66 MHz operation, all 23 inputs to the device have registers, thus pipelining the device operation. This allows external data to the synchronized, or CPU bus-oriented data to be latched. Input registers may be clocked from either of two input clock sources on either pin 2 or 3. Like all other programmable devices from Cypress, the CY7C330 is UV-light erasable, and comes in either a windowed ceramic package or in a plastic DIP or PLCC.

This application note contains four design examples; a high-speed Up/Down Counter with Limits, a 16x16 Crossbar Switch, a Pipelined Buffer, a simple Toggle Counter, and an Internal Product term numbering chart. All example source code is in Cypress PLD Toolkit^{TM} syntax.

Overview

An easy way to picture the CY7C330 is with the block diagrams in *Figure 1*. On the input side of the CY7C330 (pins 1-7 and 9-14) are 11 input registers and 3 clocks. Pin 1 is the State Clock. Each of the 11 input registers is edge-triggered, and each can use either device pin 2

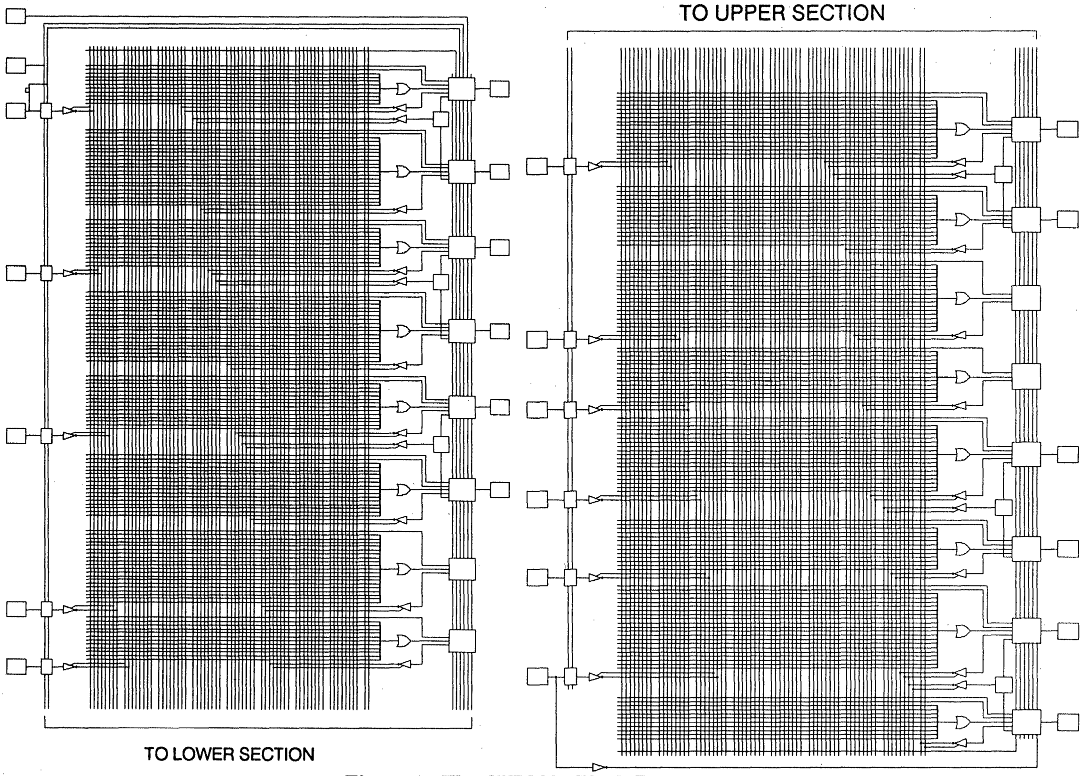


Figure 1. The CY7C330 Block Diagram

(clock 1) or pin 3 (clock 2) (shown in *Figure 2*) as a clock. An architecture bit for each input register controls the selection of the input clock. This approach allows input data to be synchronized to a clock edge, or to be loaded into the device from a CPU data bus, with the clocks being decoded I/O write signals. The setup and hold times are very short allowing high-system throughput. The outputs of these registers feed the "AND-OR-XOR" array. Pin 14 has an additional function to the input register, it can be used as a fast, asynchronous output enable to the device, allowing a CPU to read out data in the state machine registers onto a bus, for example.

On the I/O side of the device, (pins 15-20 and 23-28) are 12 macrocells. Each I/O macrocell, *Figure 3*, contains a type D register, an input register with clock controls, and output enable resources. Architecture bits for feedback selection, output enable configuration and input register clock selection allow each of the macrocells to be independently configured. Each adjacent I/O macrocell shares an input multiplexer (see *Figure 5*) allowing either macrocell register to be buried while the I/O pin is used as an input. In addition, there are four buried register macrocells (see *Figure 4*) providing additional state registers but without direct output connections.

Logic Array

The "AND-OR-XOR" array in *Figure 1* has 66 inputs and 244 product terms driving 16 "OR-XOR" gates. The 16 OR gates have from 9 to 19 inputs (variable product terms) allowing very complex designs to fit into each stage. An XOR product term for each OR output allows equations to be solved either with D or T type flip-flops in the output stage, or for active high or active low equations. Twelve product terms provide the output enable function. A global reset and preset is also generated out of the array. Each product term forms an AND function with up to 66 inputs. The 66 inputs are the true and complement signals of 33 internal nodes in the CY7C330.

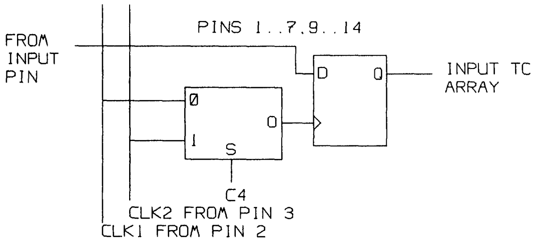


Figure 3. The CY7C330 Input Macrocell

Macrocell State Registers

The OR-XOR gates feed into 16 state registers *Figures 3 & 5*. These are edge-triggered D flip-flops with pin 1 as clock. Output from these state registers are fed back into the array allowing high-speed state machines to be constructed. Total feedback time period from Q to D and array delay from input register to state register is 15 ns, allowing a full usable clock rate of 66 MHz. Four of these registers are always buried inside the device. A buried register allows intermediate states or other functions to be built without loading an I/O pin. Of the twelve remaining registers, up to 6 can be buried, giving a total of 10 maximum usable buried registers while allowing the 28-pin device to have 17 dedicated input pins, plus 6 I/O pins, plus many other combinations. Valid I/O macrocell configurations are shown in *Figure 6*.

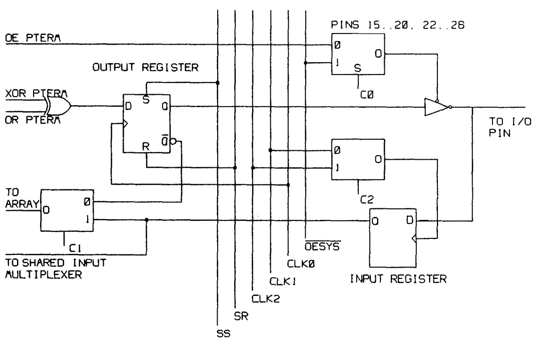


Figure 2. The CY7C330 I/O Macrocell

Additional Input Registers

Each I/O macrocell (pins 15-20 and 22-28) also has an input, edge triggered register with either pin 2 or pin 3 as clock. The total register count is 39: 16 state registers and 23 input registers.

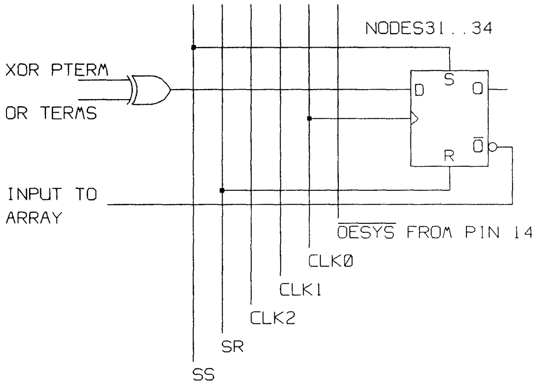


Figure 4. CY7C330 Buried Macrocell

In order to keep the device speed as high as possible, the number of inputs to the array was limited to 33 (x2) - six of the array inputs from the I/O Macrocells are multiplexed (shared). Thus three feedbacks are provided for the two output and two input registers for each set of two I/O pins. The easiest way to understand the net result is that the maximum number of buried registers in the twelve I/O Macrocells is six. Output registers that have no feedback to the array are useful for data outputs or single clock delayed Mealy outputs from the state machine.

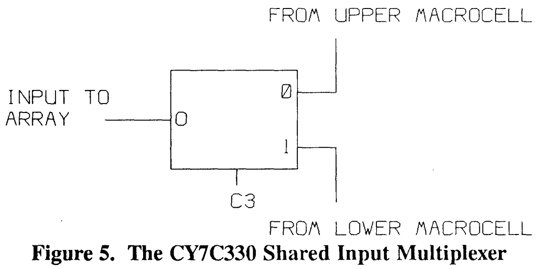


Figure 5. The CY7C330 Shared Input Multiplexer

The twelve macrocells have 24 registers total and 18 feedbacks. The assignment of functions in the user's application to physical pins in the device needs to be done with consideration of the number of feedbacks available (and the number of product terms required).

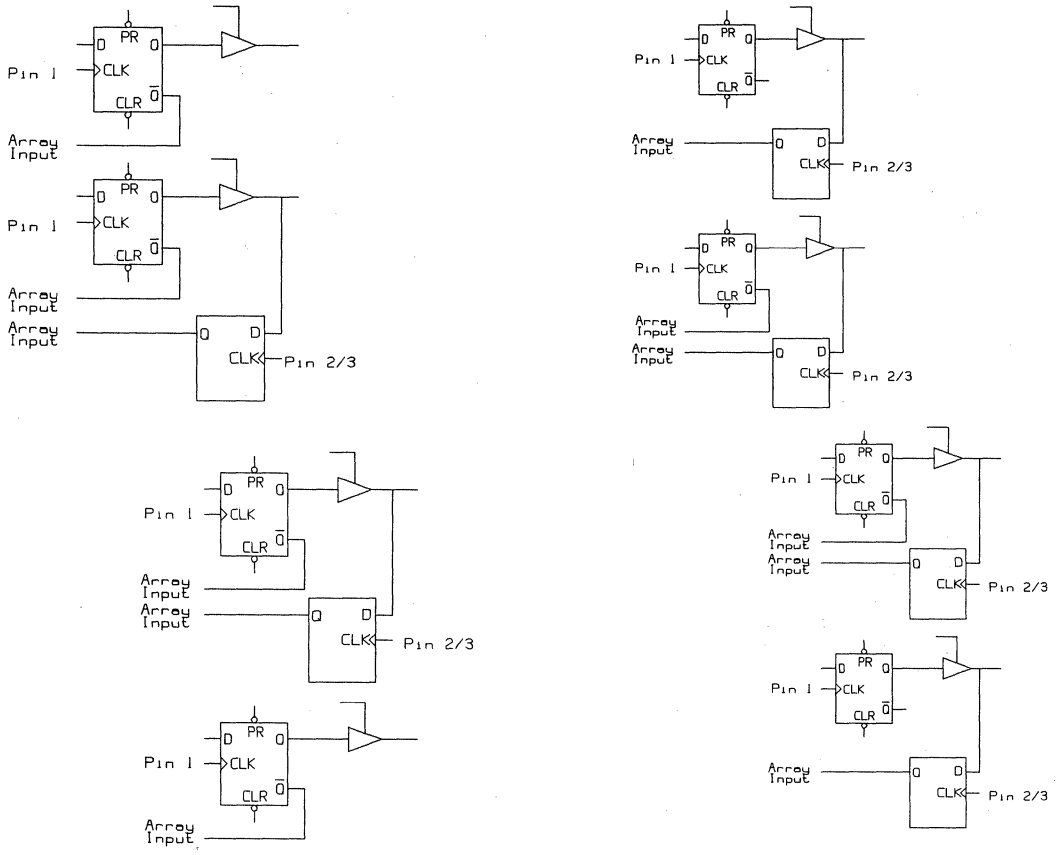


Figure 6. Four CY7C330 I/O Macrocell Configurations

Center Pinning

All Cypress CY7C330 family products use center pins for V_{CC} and V_{SS} connections. In addition, the V_{SS} for the internal logic and the V_{SS} for the output drivers are on different pins. Center power pins eliminate noise generated by both TTL and CMOS devices. This noise is inductive noise proportional to the package lead inductance. Moving the power pins to the center lowers pin inductance and noise by a factor of 3 compared with corner-pin power connections.

Splitting ground lines between input and logic on pin 8 and output drivers on pin 21 has additional benefits. Ground bounce noise is caused when outputs switch from HIGH to LOW. The more pins switching at the same time, the more noise generated. Several hundred mV can be induced on the chip's internal ground from this effect. While the level is low enough to meet output V_{ol} specs, this voltage must be considered when designing the input buffers on a chip, since it will influence the V_{il} spec of 0.8 V. 400 millivolts of ground bump noise will shift the AC effective V_{il} to 1.2 V.

By separating the input reference ground from the output ground where the noise is generated, Cypress can design a faster input buffer, because ground noise compensation is lowered or eliminated. Externally, the two grounds are connected together. Also, by placing the V_{CC} pin close to the GND pin, external 0.1 uF capacitors (as usual, one per chip) can be very close to the actual device power pins.

All Cypress EPLDs permit the registers to be preloaded into any configuration. This can vastly reduce the test time, and allows all patterns programmed into an EPLD to be completely tested. Without preload, for example, testing a multibit counter that has no reset product term could be very slow or impossible.

CY7C33X Family Technology Characteristics

The CY7C330 and most other new Cypress products are being built in the Cypress 0.8 micron, N-well CMOS, high-speed technology. New Cypress EPLDs use a dual metal layer connection method to further increase speed. This technology allows static RAMs to be built with 7 ns access times, 35 MHz FIFOs, a 33 MHz RISC processor and many other high-performance products.

Cypress uses an EPROM (vs. fuse link or EEPROM) technology for all its EPLDs (and (E)PROMS) because of the tremendous increase in manufacturing yields it offers, as well as 100% testability. This UV-erasable EPROM technology offers proven data retention, testability, and manufacturability. In addition, the Cypress 2T (2 transistor) cell design allows very high speed circuits to be built. Cypress uses this 2T cell design for performance. One transistor is used only for programming and the other for reading with each optimized for only one function. The program transistor can be larger and slower. It is designed to withstand 15 V source to drain, and is the maximum program charge on the floating gate. The read transistor can be very small and fast. Because the read bit line is only switching between 0-5 V, the sense amp is smaller and faster, and no high-current 15 V driver MOSFETS are present. The result is very fast (sub 10 ns) array times.

All Cypress devices offer protection against static discharge (ESD). This means the devices are no more sensitive than bipolar devices. By using a unique -3V substrate bias generator (Vbb), Cypress devices are protected from latchup caused by transient voltages below ground, which are commonly seen in TTL systems. This internally generated Vbb also allows the device to maintain high speed over a wide temperature range by controlling switching thresholds. No current flows in an input even under extreme undershoot situations, and there is no recovery time required for the input transistor after an undershoot.

In addition to Substrate bias for latchup elimination, Cypress uses a Stacked TTL output driver, removing the Pin to P channel transistor connection, a major source of latchup. Overshoot and noise generation is also improved by reducing the energy in HIGH to LOW transitions. Virtually all high-performance systems using TTL or CMOS adhere to the TTL standard voltage specification -- 2.0 V for a TTL HIGH and 0.8 V for a TTL LOW. This means that a P-channel output transistor for pulling the output to Vcc causes more problems than it solves because it overdrives the output. The lower voltage output from a stacked N channel output drive of 3.5 V vs. 5.0 V causes less noise on the HIGH to LOW transition because less energy needs to be switched.

Cypress uses stacked N-channel transistors on the outputs of all devices, eliminating latchup and fast transition to an overly high output "1" level. The devices are more compatible with the TTL devices Cypress replaces.

Resource Planning

Planning the assignment of functions to pins in the CY7C330 is an important step in a CY7C330 design. The resource planning sheet on the following page will be helpful for this procedure. Examples of its use are included with each application.

The decision on which pin to use is based on:

1. Asynchronous output enable, set to pin 14 or synchronous enable with a product term

2. State clock is pin 1

3. Input clock is pin 2

4. Second input clock is pin 3 or use pin 3 as a normal input if pin 2 will be the only input clock

5. Input only on pins 4-7 and 9-13

6. Device outputs: Assign pins in the sequence of counter MSB to LSB bits Pins 20, 23, 19, 24, 17, 26, 15, 28, 16, 27, 18, 25.7.

7. Use of Hidden Registers
   a. Four registers H1 to H4 are always hidden.
   b. Up to six additional hidden registers can be defined. We suggest this sequence : 25, 18, 27, 16, 23, 20;
   c. Assign input names to these six registers that are

defined. We suggest this sequence : 25, 18, 27, 16, 23, 20;

c. Assign input names to these six registers that are different from the physical device pin names;

d) The optionally hidden registers can be viewed if their output enable is made active (and the external logic driving the pin is in a high-impedance state), otherwise the OE (Output Enable) product term of the hidden register must be set to "ZERO". (NAME.ENA = 0;)

8. The remaining visible registers can still be used in applications where both inputs of a macrocell pair are used. However, one of the output registers of each adjacent pair cannot have a feedback; it is used only as an output synchronized by the State Clock on pin 1.

If, after this assignment, the compiler or assembler complains that not enough product terms are available, then some pins may have to be re-assigned.

Table 1. A CY7C330 Resources Planning Sheet

Project : Your project name

Pin	**Input Register Function**	**Input Register Clock**	**Register Function**	**Output Enable**	**# of PTerms**
1	State Clk				
2	Clk 1				
3	Input/Clk 2	1 if Input			
4	Input	1/2			
5	Input	1/2			
6	Input	1/2			
7	Input	1/2			
8	VSS				
9	Input	1/2			
10	Input	1/2			
11	Input	1/2			
12	Input	1/2			
13	Input	1/2			
14	Input/OE	1/2 if Input			
15	Input	1/2 if Input	Output	Pin 14/Pterm	9
16	Input	1/2 if input	Output	Pin 14/Pterm	19
17	Input	1/2 if input	Output	Pin 14/Pterm	11
18	Input	1/2 if input	Output	Pin 14/Pterm	17
19	Input	1/2 if input	Output	Pin 14/Pterm	13
20	Input	1/2 if Input	Output	Pin 14/Pterm	15
21	VSS				
22	VCC				
23	Input	1/2 if input	Output	Pin 14/Pterm	15
24	Input	1/2 if input	Output	Pin 14/Pterm	13
25	Input	1/2 if input	Output	Pin 14/Pterm	17
26	Input	1/2 if input	Output	Pin 14/Pterm	11
27	Input	1/2 if input	Output	Pin 14/Pterm	19
28	Input	1/2 if input	Output	Pin 14/Pterm	9
H1	None	-	-	None	19
H2	None	-	-	None	11
H3	None	-	-	None	17
H4	None	-	-	None	13
Notes :	Input Register Clock	#1 is pin 2			
		#2 is pin 3			

See the Application Note for the meaning of the pin names.
Output Enable = 14 means the asynchronous pin 14 direct enable.
Z means the pin is never active

Software Design Tools

Logic for the CY7C330 can be compiled with a number of packages available from third party independent software vendors. These include ABEL™ V3.0 from DATA I/O® and LOG/IC™ V3.0 from ISDATA®. Cypress has developed a PLD Toolkit (CY7C3101) that can be used to design any PLD that Cypress makes. All of these are logic compilers capable of converting state machine or binary logic descriptions into a JEDEC file to program the device.

The JEDEC file is the standard interface from a software development tool to a logic programmer. See the examples section for more detail on the software tools.

Logic Programmers

The CY7C330 can be programmed today on the IBM (or compatible) QuickPro™ plug-in board, and shortly, it will be able to be programmed on DATA I/O®, STAG® and other programmers. Some software tools require the user to set "fuses" or bits in the device to enable certain functions, whereas others will set the architecture bits automatically. These bits are shown in *Table 6*. Special attention needs to be applied to bit 17070: it must be set to 1 if any input register uses a clock from pin 3. These requirements will disappear in future releases of these software packages and the bits will be set automatically.

Applications Example: Pipelined Buffer

The Pipe330 example is a two-stage pipeline that simply shifts parallel data from the inputs to the outputs (see *Figure 7*). This example shows the overall Cypress PLD Toolkit source syntax, and shows how macrocells are configured.

In the Pipe330 example, the output enable for particular macrocells is either under control of pin 14 or under control of the associated product term. The latter case is the default. To control the output enable of a macrocell with pin 14, add "NENBPT" to the list of attributes following the node assignment in the configuration section.

If NENBPT does not appear in the attribute list for a node, then the output enable is controlled by the expression that follows the construct <OE> in the equations. If <OE> is not part of the equation, the output is permanently disabled. If <OE> is present, but there is not expression following it, the output is permanently enabled.

The output registers in the CY7C330 are always clocked by pin 1. The input registers can be clocked by either pin 2 or 3. Pin 2 is the default clock, so no special attributes are required for this configuration. If you wish to clock an input register with pin 3, the attribute list for that node must contain "ICLK=3".

The resource planning sheet for the pipelined buffer is in *Table 2*, and the source code is in *Appendix A*.

Test patterns for the Pipe330 example are relatively simple but a few guidelines should not be ignored. At first, the state of the registers in the device is unknown, and all of the registers are put in a known state before any outputs are checked (non-X). Another aspect of simulation of the CY7C330 is the need to look after multiple clocks. The input and output clocks should be treated separately, since the simultaneity of clock assertion is not guaranteed in programmers (or in any real system for that matter.)

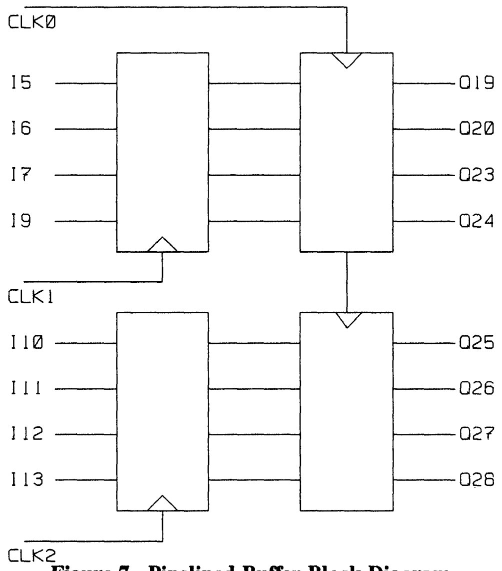


Figure 7. Pipelined Buffer Block Diagram

Table 2. Resource Planning Sheet for Pipelined Buffer

7C330 Resource Planning Sheet

Project : Pipelined Buffer

Pin	Input Register Function	Input Register Clock	Register Function	Output Enable	# of PTerms
1	State Clk				
2	Clk 1 (LHS)				
3	Clk 2 (RHS)				
4	I4	1			
5	I5	1			
6	I6	1			
7	I7	1			
8	VSS				
9	I9	1			
10	I10	2			
11	I11	2			
12	I12	2			
13	I13	2			
14	OE	-			
15	-	-	-	Z	9
16	-	-	-	Z	19
17	-	-	-	Z	11
18	-	-	-	Z	17
19	-	-	Q19	Pterm (Eqn)	13
20	-	-	Q20	Pterm (Eqn)	15
21	VSS				
22	VCC				
23	-	-	Q23	Pterm (Eqn)	15
24	-	-	Q24	Pterm (Eqn)	13
25	-	-	Q25	Pin 14	17
26	-	-	Q26	Pin 14	11
27	-	-	Q27	Pin 14	19
28	-	-	Q28	Pin 14	9
H1	None	-	-	None	19
H2	None	-	-	None	11
H3	None	-	-	None	17
H4	None	-	-	None	13

Notes : Input Register Clock
#1 is pin 2
#2 is pin 3
See the Application Note for the meaning of the pin names.
Output Enable = 14 means the asynchronous pin 14 direct enable.
Z means the pin is never active

Applications Example: 4-Bit Up/Down Toggle Counter with Preloads

The Tog330 example shows how the XOR product terms can be used to emulate a T-type flip-flop. The statement:

```
Q =    <XSUM> Q
       <SUM> T;
```

causes the XOR product term to be programmed with the feedback of the register output, making the register into a T-type. By architecture, all of the outputs are active LOW, so the T-type register configuration is active LOW. You can also use the configuration above, with the following relation:

T = J!Q + KQ

to emulate a JK-type flip-flop.

Table 3. Resource Planning Sheet for Toggle Counter

7C330 Resource Planning Sheet

Project : 4 Bit Toggle Counter

Pin	Input Register Function	Input Register Clock	Register Function	Output Enable	# of PTerms
1	State Clk				
2	Clk 1				
3	Clear	1			
4	-				
5	-				
6	-				
7	-				
8	VSS				
9	-				
10	-				
11	-				
12	-				
13	-				
14	-				
15	-	-	!Q0	Pterm	9
16	-	-	!Q1	Pterm	19
17	-	-	!Q2	Pterm	11
18	-	-	!Q3	Pterm	17
19	-	-		Z	13
20	-	-		Z	15
21	VSS				
22	VCC				
23	-	-		Z	15
24	-	-		Z	13
25	-	-		Z	17
26	-	-		Z	11
27	-	-		Z	19
28	-	-		Z	9
H1	None	-	-	None	19
H2	None	-	-	None	11
H3	None	-	-	None	17
H4	None	-	-	None	13

Notes : Input Register Clock #1 is pin 2
#2 is pin 3
See the Application Note for the meaning of the pin names.
Output Enable = 14 means the asynchronous pin 14 direct enable.
Z means the pin is never active

The resource planning sheet for the toggle counter example is in *Table 3,* and the source code can be found in *Appendix B*. *Figure 8* shows the block diagram for the design.

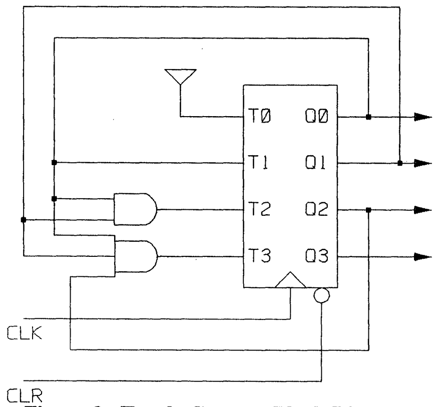


Figure 8. Toggle Counter Block Diagram

Applications Example CY7C330 Up/Down Counter with Limits

This example shows how the pins can be assigned for maximum use in the CY7C330. This counter operates at 66 MHz, counting up until the value stored in the 8-bit upper limit register is reached, then down until the lower limit is reached. Also included is a method to preload the counter to either the upper or lower limit, as well as a device reset.

Let us assume that the two 8-bit limit registers are loaded from a CPU. The lower limit is on pins 4 to 12, with a 9th bit for preload on pin 13. Clock for this lower limit is pin 2. The upper limit is loaded via pins 15-27, with pin 27 being the 9th preload bit. These pins are also used for reading out the counter value, and pin 14 is the output enable for the 8 bit up/down counter. Four buried registers are used to detect equality of the counter with the limits, to maintain up/down direction and to detect the preload request as an edge-triggered signal. By using the XOR product terms, the counter needs only 9 total products even on the most significant bit. Without XOR, the 8th bit mould needs 18 product terms because of the 2 preload sources. Because of the large number of product terms per output in the CY7C330, this counter can operate at 66 MHz.

The contents of the counter can be read out when pin 14 (direct output enable) is LOW. In a bus-oriented system, a microprocessor could read out the register if a decoded I/O read signal were applied to pin 14. Note that the other method of output enable, via the array, requires a clock edge to load the required enable input condition into the input registers. When pin 14 is high, the upper limit register can be loaded, for example from a microprocessor bus. The lower limit register can be loaded at any time. The block diagram for this design is *Figure 9.* The resource planning sheet for this design is in *Table 4* and the code is in *Appendix C.*

Description:

The device is a up-down 8-bit counter that counts between the limits stored in two registers. The operation is as follows:

Lower limit (LL) data is loaded on the positive edge of pin 2. There are 8 data bits plus 2 control bits, LPL and Reset. If LPL is low, then only the limit compare register is changed. If LPL is high, then the LL data is loaded into the counter on the next clock edge, and the counter will count up. The LL data is one count higher than the actual lower limit. If RESET is active, then all internal registers will be reset to 0 as long as the reset bit is set in the LL register.

Upper limit (UL) data is loaded on the positive edge of pin 3. There are 8 data bits plus a preload control bit. If UPL is low, then only the limit compare register is changed. If UPL is high, then the UL data is loaded into the counter on the next clock edge, and the counter will count down. UL data is multiplexed with Counter output data. The UL data is one count lower than the

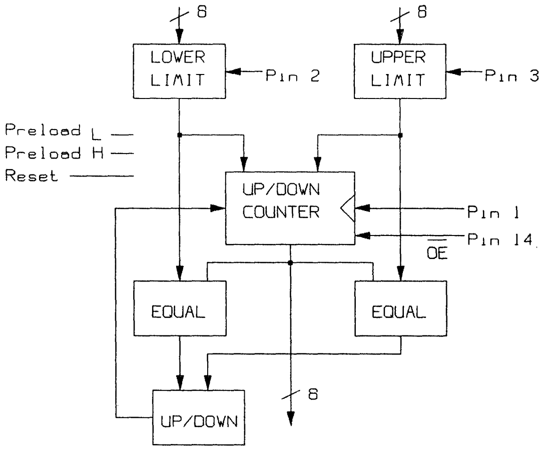


Figure 9. Up/Down Counter Block Diagram

actual upper limit. Pin 16 is the RESET input. Pin 14 is the active low output enable for the counter. The counter can be read at any time. Pin 1 is the clock for the counter. Pins 18 and 20 are connected together for data bit 6. Pins 23 and 25 are connected together for data bit 7.

The buried (hidden) registers are used as follows:

H1 is loaded with the result of the comparison between the counter and UL. H2 is UPL or LPL delayed by one clock edge. It is used as an edge detect. H3 is loaded with the result of the comparison between the counter and LL. H4, when high, forces the counter to count up.

Table 4. Resource Planning Sheet for UP/Down Counter

7C330 Resource Planning Sheet

Project : Up/Down Counter with Limits

Pin	Input Register Function	Input Register Clock	Register Function	Output Enable	# of PTerms
1	State Clk				
2	Clk 1				
3	Clk 2				
4	LL0	1			
5	LL1	1			
6	LL2	1			
7	LL3	1			
8	VSS				
9	LL4	1			
10	LL5	1			
11	LL6	1			
12	LL7	1			
13	PRELOAD LOW	1			
14	COUNTER OE	-			
15	UL1	2	CNT1	Pin 14	9
16	Reset	1	-	Z	19
17	UL3	2	CNT3	Pin 14	11
18	UL6	2	-	Z	17
19	UL4	2	CNT4	Pin 14	13
20	-	-	CNT6	Pin 14	15
21	VSS				
22	VCC				
23	-	-	CNT7	Pin 14	15
24	UL5	2	CNT5	Pin 14	13
25	UL7	2	-	Z	17
26	UL2	2	CNT2	Pin 14	11
27	PRELOAD HIGH	2	-	Z	19
28	UL0	2	CNT0	Pin 14	9
H1	None	-	Up Equals	None	19
H2	None	-	L/H Prel'Done	None	11
H3	None	-	Down Equals	None	17
H4	None	-	Up Count	None	13

Notes :Input Register Clock #1 is pin 2
#2 is pin 3
See the Application Note for the meaning of the pin names.
Output Enable = 14 means the asynchronous pin 14 direct enable.
Z means the pin is never active

Application Example: 16 x 16 Crossbar Switch

A data switch capable of multiplexing 16 inputs into 4 outputs can be built with one CY7C330. The 66 MHz clock rate allows even asynchronous input signals of up to 33 MHz to be switched through the device. The compact 300 mil package saves PCB space. Normally such a multiplexer would need at least 40 pins partitioned as follows:

16 input pins,
4 output pins,
4 x 4 = 16 selection inputs
4 pins for power and clock connections

No other PLD today can perform this function using a single device, because of the logic requirement (i.e., the number of product terms required per output) as well as the timing requirement. However, this is no problem for the CY7C330; the entire design fits in one 300 mil, 28-pin package and runs with a maximum clock rate of 66 MHz.

Description:

This example uses 12 state registers plus 4 input registers to act as the 4 x 4 bit selection registers. Each output channel needs a 4 bit register to select one of 16 input channels. In this example we construct a 4 stage, 4 bit-wide shift register inside the part to hold the select status. This way the data to these 4 x 4 bits can be loaded via only 4 pins without needing any address pins. When the PL (PRELOAD) pin 3 is LOW, input data bits 0 to 3 become the selector data lines; 5 clock pulses will shift the select data through the device into the selectors 1, 2 and 3 as well as the output pins. Setting pin 3 HIGH after the fifth pulse will load the output data pins into the select register 0. This last load operation utilizes the function of pin 3 as a data pin as well as a clock. Setting pin 3 LOW switches the internal logic from a selector into a shift register; setting pin 3 HIGH is a clock edge which loads the data output into the input registers associated with the output pins (16, 18, 25, 27).

This design requires that we "bury" the output register of several of the I/O macrocells, and use the pin as an input by utilizing a shared input mux. This is accomplished in the configuration section of the source file. First, we must assign the name of the output register to the macrocell node number. Since the default configuration is for the Q output of the output register to be fed back into the array, no other configuration attributes are needed here. The name of the input is assigned to the node number of the shared input mux adjacent to the pin. The default for the shared input muxes is to pass the data on the even pin into the array. If the input is to come from an odd numbered pin, you must add the attribute "SRC=N" (where *N* is the pin number) to the list of attributes in parentheses following the node name. For an example of this syntax, refer to d10 and sa2 in the source file.

The space advantage of the CY7C330 in this crossbar switch application becomes especially important as the size of the matrix increases. A 32 x 32 matrix would need only 16 devices vs. 64 PALC22V10s or 96 TTL circuits. Loading of the internal data selection registers is easily done with a Cypress 24-pin EPLD, the PLDC20G10, and a FIFO. A CPU would load the 16 x 4 bit selector information into the FIFO and the PLDC20G10 would move the data from the FIFO into the device. One PLDC20G10 and one 16 x 4 (or larger) FIFO is required. The Cypress CY7C403 would be an ideal FIFO for this application.

The resource planning sheet for the 16 X 16 crossbar switch design is in *Table 5*, and a block diagram of the design is pictured in *Figure 10*. The source code can be found in *Appendix D*.

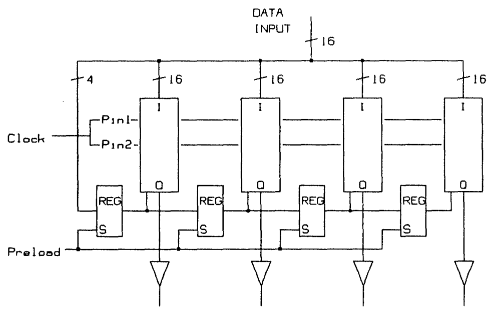


Figure 10. 16X16 Crossbar Switch Block Diagram

Reading the JEDEC Map CY7C330 Internal Array Reference

Table 6 is intended to help read the JEDEC MAP of a CY7C330. The pin or node reference number is on the left. These numbers correspond to the pin and node numbers on the block diagram *Figure 1.* The column labeled "Input True" gives the sequential number (left to right) of the column corresponding to the non-inverted input to the array. If the number is even, then the false input is the next-higher integer; if the number is odd, then the false input is the next lower integer. The number of product terms in each output stage is listed, along with the JEDEC offset (sequential fuse position) for each.

Table 5. Resource Planning Sheet for Crossbar Switch

7C330 Resources Planning Sheet
Project :16 X 16 Crossbar Switch

Pin	Input Register Function	Input Register Clock	Register Function	Output Enable	# of PTerms
1	State Clk				
2	Clk 1				
3	Sel PRELOAD	1			
4	Data 0	1			
5	Data 1	1			
6	Data 2	1			
7	Data 3	1			
8	VSS				
9	Data 4	1			
10	Data 5	1			
11	Data 6	1			
12	Data 7	1			
13	Data 8	1			
14	Data 9	1			
15	Data 10	1	Select A2	Z	9
16	Select D0	2	Output 3	Pterm	19
17	Data 11	1	Select A1	Z	11
18	Select C0	2	Output 2	Pterm	17
19	Data 12	1	Select C1	Z	13
20	-	1	Select D1	Z	15
21	VSS				
22	VCC				
23	-	1	Select B2	Z	15
24	Data 13	1	Select A2	Z	13
25	Select B0	2	Output 1	Pterm	17
26	Data 14	1	Select C2	Z	11
27	Select A0	2	Output 0	Pterm	19
28	Data 15	1	Select D2	Pterm	9
H1	None	-	Select A3	None	19
H2	None	-	Select B3	None	11
H3	None	-	Select C3	None	17
H4	None	-	Select D3	None	13

Notes : Input Register Clock #1 is pin 2
#2 is pin 3
See the Application Note for the meaning of the pin names.
Output Enable = 14 means the asynchronous pin 14 direct enable.
Z means the pin is never active

Table 6. The CY7C330 Internal Array Reference List

Pin or Node	Function	Input True	# of Pterms	OE	XOR	1st OR
1	State Clock					
2	Input Clock1					
3	Input Clock2	0				
4	Input Register	2				
5	Input Register	4				
6	Input Register	6				
7	Input Register	8				
8	VSS					
9	Input Register	10				
10	Input Register	12				
11	Input Register	14				
12	Input Register	16				
13	Input Register	18				
14	Input Register	20				
15	I/O Regs, mux	65	9	L16236	L16302	L16368
N-35	mux input(node)	62				
16	I/O Regs, mux	61	19	L14850	L14916	L14982
17	I/O Regs, mux	59	11	L13992	L14058	L14124
N-36	mux input(node)	56				
18	I/O Regs, mux	55	17	L12738	L12804	L12870
19	I/O Regs, mux	49	13	L9636	L9702	L9768
N-37	mux input(node)	46				
20	I/O Regs, mux	45	15	L8514	L8580	L8646
21	VSS					
22	VCC					
23	I/O Regs, mux	39	15	L5280	L5346	I 5412
N-38	mux input(node)	36				
24	I/O Regs, mux	35	13	L4290	L4356	L4422
25	I/O Regs, mux	33	17	L3036	L3102	L3168
N-39	mux input(node)	30				
26	I/O Regs, mux	29	11	L2178	L2244	L2310
27	I/O Regs, mux	27	19	L792	L858	L914
N-40	mux input(node)	24				
28	I/O Regs, mux	23	9	L66	L132	L198
N-29	Sync. Reset			L0		
N-30	Sync. Preset			L16962		
N-31	Buried Register	40	13		L11814	L11870
N-32	Buried Register	42	17		L10626	L10692
N-33	Buried Register	50	11		L7722	L7788
N-34	Buried Register	52	19		L6402	L6468

Appendix A. PLD ToolKit Source Code for Pipelined Buffer

```
CY7C330;                         {Pipe330}

CONFIGURE;

CkS (node=1),                    {Output register clock}
Ck1,                             {Input register clock 1}
Ck2,                             {Input register clock 2}
I0 (iclk=3),                     {Input 0, clocked by Ck2 (pin 3)}
I1 (iclk=3),                     {Input 1, clocked by Ck2 (pin 3)}
I2 (iclk=3),                     {Input 2, clocked by Ck2 (pin 3)}
I3 (iclk=3),                     {Input 3, clocked by Ck2 (pin 3)}
I4 (node=9),                     {Input 4, clocked by Ck1 (pin 2)}
I5,                              {Input 4, clocked by Ck1 (pin 2)}
I6,                              {Input 4, clocked by Ck1 (pin 2)}
I7,                              {Input 4, clocked by Ck1 (pin 2)}
OE1,                             {output enable for Q<7:4>}
!OE2(node=14),                   {direct output enable for Q<7:0>}
Q7,                              {Output 7, clocked by CkS, enabled by OE1&!OE2}
Q6,                              {Output 6, clocked by CkS, enabled by OE1&!OE2}
Q5,                              {Output 5, clocked by CkS, enabled by OE1&!OE2}
Q4,                              {Output 4, clocked by CkS, enabled by OE1&!OE2}
Q3(nenbpt),                      {Output3, clocked by CkS, enabled: pin14}
Q2(nenbpt),                      {Output2, clocked by CkS, enabled: pin14}
Q1(node=23,nenbpt),              {Output1, clk: CkS, OE:: pin14}
Q0(nenbpt),                      {Output0, clocked by CkS, enabled: pin14}
!RST(iop),                       {low asserted reset, I/O macrocell as input}
reset(node=29),                  {internal reset node}

EQUATIONS;

reset = RST;

!Q0 =  <sum> !I0;

!Q1 =  <sum> !I1;

!Q2 =  <sum> !I2;

!Q3 =  <sum> !I3;

!Q4 =  <oe> OE1 & OE2
       <sum> !I4;

!Q5 =  <oe> OE1 & OE2
       <sum> !I5;

!Q6 =  <oe> OE1 & OE2
       <sum> !I6;

!Q7 =  <oe> OE1 & OE2
       <sum> !I7;
                                 {end of file}
```

Appendix B. PLD ToolKit Source Code for a Toggle Counter

```
CY7C330;                          {Tog330}

CONFIGURE;

CkS,                              {Count clock, This is pin1 since it is first in the list.}
Ck1,                              {Input clock, This is pin2 since it is next.}
!Clr,                             {Low true clear, Pin3 is next in sequential order.}
!OE(node=14),                     {Low asserted output enable pin, pin 14}
!Q0(nenbpt),                      {Q0-Q3 are the counter outputs - pins 15-18.}
!Q1(nenbpt),
!Q2(nenbpt),
!Q3(nenbpt),
reset(node=29),                   {The reset product term is node 29.}

EQUATIONS;

reset = Clr;

Q0 =    <xsum> Q0                 {Feeding the register output back into the XOR emulates a T flop.}
        <sum>;                    {T input - No expression after the connective <sum> means always asserted}

Q1 =    <xsum> Q1                 {Feeding the register output back into the XOR emulates a T flop.}
        <sum> Q0;                 {T input}

Q2 =    <xsum> Q2                 {Feeding the register output back into the XOR emulates a T flop.}
        <sum> Q1 & Q0;            {T input}

Q3 =    <xsum> Q3                 {Feeding the register output back into the XOR emulates a T flop.}
        <sum> Q2 & Q1 & Q0;       {T input}

                                  {end of  file}
```

Appendix C. PLD ToolKit Source Code for Up/Down Counter

```
CY7C330;                                        {File: COUNTER.CYP Date: 11/9/1988 }
CONFIGURE;
CLK (node=1), LLC (node=2), ULC (node=3),       {Count cLocK, Lower Limit Clock, Upper Limit Clock}
LL0 (node=4, iclk=2), LL1, LL2, LL3,            {The Lower Limit register is clocked by pin 2-LLC- by default.}
LL4 (node=9), LL5, LL6, LL7,                    {The register is located at pins 4-7, 9-12 - pin 8 is Vss.}
LPL(node=13),                                   {Lower limit PreLoad}
/CNTOE (node=14),                               {Counter output enable on pin 14}
CNT0 (node=28, nenbpt,oclk=1, iclk=3),          {The counter itself is in the output register of various I/O macrocells}
CNT1 (node=15, nenbpt, iclk=3),                 {as noted in the node numbers after the names.Pin 1 always clocks the}
CNT2 (node=26, nenbpt, iclk=3),                 {output registers-oclk=1 was included once for documentation.}
CNT3 (node=17, nenbpt, iclk=3),                 {'nenbpt' specifies that the output enable is controlled by pin 14}
CNT4 (node=19, nenbpt, iclk=3),                 {rather than the output enable product terms in each macrocell}
CNT5 (node=24, nenbpt, iclk=3),                 {Most of these macrocells will be bidirectional, with the Upper Limit}
CNT6 (node=20, nenbpt),                         {register residing in the input registers. 'iclk=3' specifies that pin 3 }
CNT7 (node=23, nenbpt),                         {clocks the input registers. This overrides the default, pin 2. }
                                                {The output register is fed back into array by default.}
UL0 (node=40, src=28),                          {UL0 is the input reg of pin28, routed thru shared input mux-node40}
UL1 (node=35, src=15),                          {UL1 is the input reg of pin15, routed thru shared input mux-node35}
UL2 (node=39, src=26),                          {UL2 is the input reg of pin26, routed thru shared input mux-node39}
UL3 (node=36, src=17),                          {UL3 is the input reg of pin17, routed thru shared input mux-node36}
UL4 (node=37, src=19),                          {UL4 is the input reg of pin19, routed thru shared input mux-node37}
UL5 (node=38, src=24),                          {UL5 is the input reg of pin24, routed thru shared input mux-node38}
UL6 (node=18, iop, iclk=3),                     {UL6 is the input reg of pin18, 'iop' selects array input from input reg}
UL7 (node=25, iop, iclk=3),                     {UL7 is the input reg of pin25, 'iop' selects array input from input reg}
UPL (node=27, iop, iclk=3),                     {Upper limit PreLoad, array input from input reg, clocked by pin 3}
/reset (node=16, iop),                          {low asserted clear, array input from input reg, clocked by pin 2}
node29 (node=29),                               {The reset product term is node29}
UP (node=31),                                   {buried node 31 selects the counter direction, clocked by pin 1}
LEQUAL (node=32),                               {buried node 32 compares counter with lower limit, clocked by pin 1}
PLDONE (node=33),                               {buried node 33 is the preload done flag, clocked by pin 1}
UEQUAL (node=34),                               {buried node 34 compares counter with upper limit, clocked by pin 1}

EQUATIONS;

/CNT0=<XSUM> /CNT0
        <SUM> /LPL  & /UPL
        <SUM> /PLDONE
        <SUM> /LL0  & LPL  & CNT0
        <SUM> /CNT0 & UL0 & UPL
        <SUM> LL0  & LPL  & /CNT0
        <SUM> CNT0 & /UL0 & UPL;

/CNT1=<XSUM> /CNT1
        <SUM> /LPL & CNT0 & /UPL & /UP
        <SUM> /LPL & /CNT0 & /UPL & UP
        <SUM> /LL1 & LPL & PLDONE & CNT1
        <SUM> LL1 & LPL & PLDONE & /CNT1
        <SUM> UPL & PLDONE & /UL1 & CNT1
        <SUM> UPL & PLDONE & UL1 & /CNT1
        <SUM> CNT0 & /PLDONE & /UP
        <SUM> /CNT0 & /PLDONE & UP;
```

Appendix C. Source Code for Up/Down Counter (continued)

```
/CNT2=<XSUM> /CNT2
        <SUM> /LPL  & CNT0 & /UPL & /UP & CNT1
        <SUM> /LPL  & /CNT0 & /UPL & UP & /CNT1
        <SUM> /LL2  & LPL  & CNT2 & PLDONE
        <SUM> LL2  & LPL  & /CNT2 & PLDONE
        <SUM> UPL & CNT2 & /UL2 & PLDONE
        <SUM> UPL & /CNT2 & UL2 & PLDONE
        <SUM> CNT0 & /PLDONE & /UP & CNT1
        <SUM> /CNT0 & /PLDONE & UP & /CNT1;

/CNT3=<XSUM> /CNT3
        <SUM>/LPL&CNT0&/UPL&CNT2&/UP&CNT1
        <SUM>/LPL&/CNT0&/UPL&/CNT2&UP&/CNT1
        <SUM> /LL3 & LPL & PLDONE & CNT3
        <SUM> LL3 & LPL & PLDONE & /CNT3
        <SUM> UPL & PLDONE & /UL3 & CNT3
        <SUM> UPL & PLDONE & UL3 & /CNT3
        <SUM>CNT0&CNT2&/PLDONE&/UP&CNT1
        <SUM>/CNT0&/CNT2&/PLDONE&UP&/CNT1;

/CNT4=<XSUM> /CNT4
        <SUM> /LL4 & LPL & PLDONE & CNT4
        <SUM> LL4 & LPL & PLDONE & /CNT4
        <SUM> UPL & PLDONE & /UL4 & CNT4
        <SUM> UPL & PLDONE & UL4 & /CNT4
        <SUM> /LPL & CNT0 & /UPL & CNT2 & /UP  & CNT3 & CNT1
        <SUM> /LPL & /CNT0 & /UPL & /CNT2 & UP & /CNT3 & /CNT
        <SUM> CNT0 & CNT2 & /PLDONE & /UP & CNT3 & CNT1
        <SUM> /CNT0 & /CNT2 & /PLDONE & UP & /CNT3 & /CNT1;

/CNT5=<XSUM> /CNT5
        <SUM> /LL5  & LPL  & CNT5 & PLDONE
        <SUM> LL5  & LPL  & /CNT5 & PLDONE
        <SUM> UPL & CNT5 & /UL5 & PLDONE
        <SUM> UPL & /CNT5 & UL5 & PLDONE
        <SUM> /LPL  & CNT0 & /UPL & CNT2 & CNT4  & /UP & CNT3 & CNT1
        <SUM> /LPL  & /CNT0 & /UPL & /CNT2 & /CNT4 & UP & /CNT3 & /CNT1
        <SUM> CNT0 & CNT2 & /PLDONE & CNT4 & /UP & CNT3 & CNT1
        <SUM> /CNT0 & /CNT2 & /PLDONE & /CNT4 & UP  & /CNT3 & /CNT1;

/CNT6=  <XSUM> /CNT6
        <SUM> /LL6  & LPL  & PLDONE & CNT6
        <SUM> LL6  & LPL  & PLDONE & /CNT6
        <SUM> UPL & PLDONE & CNT6 & /UL6
        <SUM> UPL & PLDONE & /CNT6 & UL6
        <SUM>/LPL&CNT0&/UPL&CNT2&CNT5&CNT4 & /UP & CNT3 & CNT1
        <SUM> /LPL & /CNT0 & /UPL & /CNT2 & /CNT5  & /CNT4 & UP & /CNT3 & /CNT1
        <SUM>CNT0&CNT2&CNT5&/PLDONE&CNT4 & /UP & CNT3 & CNT1
        <SUM> /CNT0 & /CNT2 & /CNT5 & /PLDONE & /CNT4 & UP & /CNT3 & /CNT1;
```

Appendix C. Source Code for Up/Down Counter (continued)

```
/CNT7=<XSUM> /CNT7
        <SUM> /LL7  & LPL  & CNT7 & PLDONE
        <SUM> LL7  & LPL  & /CNT7 & PLDONE
        <SUM> UPL & /UL7 & CNT7 & PLDONE
        <SUM> UPL & UL7 & /CNT7 & PLDONE
        <SUM> /LPL & CNT0 & /UPL & CNT2 & CNT5 & CNT6 & CNT4 & /UP & CNT3 & CNT1
        <SUM> /LPL & /CNT0 & /UPL & /CNT2 & /CNT5 & /CNT6 & /CNT4 & UP & /CNT3 & /CNT1
        <SUM> CNT0 & CNT2 & CNT5 & /PLDONE & CNT6 & CNT4 & /UP & CNT3 &CNT1
        <SUM> /CNT0 & /CNT2 & /CNT5 & /PLDONE & /CNT6 & /CNT4 & UP & /CNT3 & /CNT1;

node29 = <SUM> reset;

UP=<XSUM> UP
        <SUM> /UEQUAL & /UP
        <SUM> /LEQUAL & UP
        <SUM> UPL & PLDONE & /UP
        <SUM> LPL  & PLDONE & UP;

PLDONE=<SUM> /LPL  & /UPL;

LEQUAL=<SUM> LL6  & /CNT6
        <SUM> /LL7  & CNT7
        <SUM> LL7  & /CNT7
        <SUM> LL3  & /CNT3
        <SUM> /LL5  & CNT5
        <SUM> LL5  & /CNT5
        <SUM> /LL1  & CNT1
        <SUM> LL0  & /CNT0
        <SUM> /LL2  & CNT2
        <SUM> /LL4  & CNT4
        <SUM> LL4  & /CNT4
        <SUM> /LL0  & CNT0
        <SUM> LL1  & /CNT1
        <SUM> /LL6  & CNT6
        <SUM> /LL3  & CNT3
        <SUM> LL2  & /CNT2;

UEQUAL=<SUM> /CNT6 & UL6
        <SUM> /UL7 & CNT7
        <SUM> UL7 & /CNT7
        <SUM> UL3 & /CNT3
        <SUM> CNT5 & /UL5
        <SUM> /CNT5 & UL5
        <SUM> /UL1 & CNT1
        <SUM> /CNT0 & UL0
        <SUM> CNT2 & /UL2
        <SUM> /UL4 & CNT4
        <SUM> UL4 & /CNT4
        <SUM> CNT0 & /UL0
        <SUM>  UL1 & /CNT1
        <SUM> CNT6 & /UL6
        <SUM> /UL3 & CNT3
        <SUM> /CNT2 & UL2;
```

Appendix D. PLD ToolKit Source Code for Crossbar Switch

```
CY7C330;                                                {16X16 Crossbar Swithch}

configure;
clk (node=1), iclk, pl,
d0, d1, d2, d3,
d4 (node=9), d5, d6, d7, d8, d9,
d10 (node=35, src=15), d11 (node=36, src=17),
d12 (node=37, src=19), d13 (node=38, src=24),
d14 (node=39, src=26), d15 (node=40, src=28),
sa1 (node=17), sa2 (node=15), sa3 (node=34),
sb1 (node=24), sb2 (node=23), sb3 (node=33),
sc1 (node=19), sc2 (node=26), sc3 (node=32),
sd1 (node=20), sd2 (node=28), sd3 (node=31),
y0 (node=27, iop, iclk=3),                              {Input reg is sa0}
y1 (node=25, iop, iclk=3),                              {Input reg is sb0}
y2 (node=18, iop, iclk=3),                              {Input reg is sc0}
y3 (node=16, iop, iclk=3),                              {Input reg is sd0}

EQUATIONS;

/sa1 =  <SUM> /pl & /sa2
        <SUM> pl & /sa1;

/sa2 =<SUM> /pl & sa3
        <SUM> pl & /sa2;

sa3 =    <SUM> /pl & d0
         <SUM> pl & sa3;

/sb1=   <SUM> /pl & /sb2
        <SUM> pl & /sb1;

/sb2=   <SUM> /pl & sb3
        <SUM> pl & /sb2;

sb3 =   <SUM> /pl & d1
        <SUM> pl & sb3;

/sc1=   <SUM> /pl & /sc2
        <SUM> pl & /sc1;

/sc2 =  <SUM> /pl & sc3
        <SUM> pl & /sc2;

sc3 =   <SUM> /pl & d2
        <SUM> pl & sc3;

/sd1 =  <SUM> /pl & /sd2
        <SUM> pl & /sd1;

/sd2 =  <SUM> /pl & sd3
        <SUM> pl & /sd2;

sd3 =   <SUM> /pl & d3
        <SUM> pl & sd3;
```

Appendix D. Source Code for Crossbar Switch (continued)

```
/y3 =   <OE> /pl
        <SUM> pl & /d0 & /sa3 & /sb3 & /sc3 & /sd3
        <SUM> pl & /d1 & sa3 & /sb3 & /sc3 & /sd3
        <SUM> pl & /d2 & /sa3 & sb3 & /sc3 & /sd3
        <SUM> pl & /d3 & sa3 & sb3 & /sc3 & /sd3
        <SUM> pl & /d4 & /sa3 & /sb3 & sc3 & /sd3
        <SUM> pl & /d5 & sa3 & /sb3 & sc3 & /sd3
        <SUM> pl & /d6 & /sa3 & sb3 & sc3 & /sd3
        <SUM> pl & /d7 & sa3 & sb3 & sc3 & /sd3
        <SUM> pl & /d8 & /sa3 & /sb3 & /sc3 & sd3
        <SUM> pl & /d9 & sa3 & /sb3 & /sc3 & sd3
        <SUM> pl & /sa3 & sb3 & /sc3 & sd3 & /d10
        <SUM> pl & sa3 & sb3 & /sc3 & sd3 & /d11
        <SUM> pl & /sa3 & /sb3 & /d12 & sc3 & sd3
        <SUM> pl & /d13 & sa3 & /sb3 & sc3 & sd3
        <SUM> pl & /d14 & /sa3 & sb3 & sc3 & sd3
        <SUM> pl & /d15 & sa3.& sb3 & sc3 & sd3
        <SUM> /pl & sd1;

/y2 =   <OE> /pl
        <SUM> pl & /d0 & sd2 & sc2 & sb2 & sa2
        <SUM> pl & /d1 & sd2 & sc2 & sb2 & /sa2
        <SUM> pl & /d2 & sd2 & sc2 & /sb2 & sa2
        <SUM> pl & /d3 & sd2 & sc2 & /sb2 & /sa2
        <SUM> pl & /d4 & sd2 & /sc2 & sb2 & sa2
        <SUM> pl & /d5 & sd2 & /sc2 & sb2 & /sa2
        <SUM> pl & /d6 & sd2 & /sc2 & /sb2 & sa2
        <SUM> pl & /d7 & sd2 & /sc2 & /sb2 & /sa2
        <SUM> pl & /d8 & /sd2 & sc2 & sb2 & sa2
        <SUM> pl & /d9 & /sd2 & sc2 & sb2 & /sa2
        <SUM> pl & /sd2 & sc2 & /sb2 & /d10 & sa2
        <SUM> pl & /sd2 & sc2 & /sb2 & /d11 & /sa2
        <SUM> pl & /sd2 & /sc2 & sb2 & /d12 & sa2
        <SUM> pl & /sd2 & /sc2 & /d13 & sb2 & /sa2
        <SUM> pl & /sd2 & /sc2 & /d14 & /sb2 & sa2
        <SUM> pl & /sd2 & /d15 & /sc2 & /sb2 & /sa2
        <SUM> /pl & sc1;

/y1 =   <OE> /pl
        <SUM> pl & /d0 & sb1 & sd1 & sc1 & sa1
        <SUM> pl & /d1 & sb1 & sd1 & sc1 & /sa1
        <SUM> pl & /d2 & /sb1 & sd1 & sc1 & sa1
        <SUM> pl & /d3 & /sb1 & sd1 & sc1 & /sa1
        <SUM> pl & /d4 & sb1 & sd1 & /sc1 & sa1
        <SUM> pl & /d5 & sb1 & sd1 & /sc1 & /sa1
        <SUM> pl & /d6 & /sb1 & sd1 & /sc1 & sa1
        <SUM> pl & /d7 & /sb1 & sd1 & /sc1 & /sa1
        <SUM> pl & /d8 & sb1 & /sd1 & sc1 & sa1
        <SUM> pl & /d9 & sb1 & /sd1 & sc1 & /sa1
        <SUM> pl & /sb1 & /sd1 & sc1 & sa1 & /d10
        <SUM> pl & /sb1 & /sd1 & sc1 & /d11 & /sa1
        <SUM> pl & sb1 & /sd1 & /d12 & /sc1 & sa1
        <SUM> pl & sb1 & /d13 & /sd1 & /sc1 & /sa1
        <SUM> pl & /d14 & /sb1 & /sd1 & /sc1 & sa1
        <SUM> pl & /d15 & /sb1 & /sd1 & /sc1 & /sa1
        <SUM> /pl & sb1;
```

Appendix D. Source Code for Crossbar Switch (continued)

```
/y0 =   <OE> /pl
        <SUM> pl & /d0 & /y0 & /y1 & /y2 & /y3
        <SUM> pl & /d1 & y0 & /y1 & /y2 & /y3
        <SUM> pl & /d2 & /y0 & y1 & /y2 & /y3
        <SUM> pl & /d3 & y0 & y1 & /y2 & /y3
        <SUM> pl & /d4 & /y0 & /y1 & y2 & /y3
        <SUM> pl & /d5 & y0 & /y1 & y2 & /y3
        <SUM> pl & /d6 & /y0 & y1 & y2 & /y3
        <SUM> pl & /d7 & y0 & y1 & y2 & /y3
        <SUM> pl & /d8 & /y0 & /y1 & /y2 & y3
        <SUM> pl & /d9 & y0 & /y1 & /y2 & y3
        <SUM> pl & /y0 & y1 & /y2 & y3 & /d10
        <SUM> pl & y0 & y1 & /y2 & /d11 & y3
        <SUM> pl & /y0 & /y1 & /d12 & y2 & y3
        <SUM> pl & y0 & /y1 & /d13 & y2 & y3
        <SUM> pl & /y0 & /d14 & y1 & y2 & y3
        <SUM> pl & /d15 & y0 & y1 & y2 & y3
        <SUM> /pl & sa1;
```

CYPRESS SEMICONDUCTOR

CY7C330 State Machine Example: SCSI Host Adapter

Introduction

This application note describes a minimal, though extremely fast, SCSI (Small Computer Systems Interface) controller that is built up from a few parts surrounding a CY7C330 synchronous state machine PLD. The controller is compliant with the SCSI standard for a host-based minimally featured interface.

A optimal speeds are achieved by efficiently using various features of the CY7C330. The 66 MHz speed, the input registers, and the device size -- including the array size -- are all features that help to achieve this level of performance.

At 66 Mhz the register to register transfers can occur at 15 ns intervals, which is fast enough to keep datapath transfers out of the way of SCSI transfers. In order to achieve optimal throughput, the SCSI handshake transfer must be made the limiting factor, so this clock speed is necessary.

The input registers are used to synchronize external signals. Synchronization is necessary so that the state machine can respond to these signals, and the input section of the state machine is the correct place to perform the task. Since the signals are synchronized at the input to the array, adherence to Gray code transitions can be ignored in the design, and thus time critical transitions can be made in less cycles.

The device and array size of the CY7C330 are sufficient to accommodate the entire control section of the interface. In fact, because the device is large enough, several signals are shared, and therefore more features can be accommodated in this design than would be the case if the interface was constructed from smaller PLDs.

The minimally featured SCSI Host implementation is a complete interface to one or more SCSI controllers from a single host.

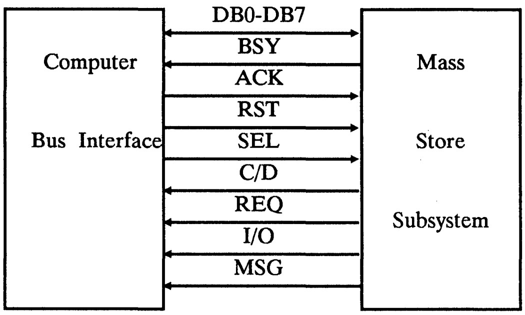


Figure 1. Small Computer System Interface (SCSI)

Conventions

In this document, conventions are followed so that signal names in timing and state diagrams can be related to schematics without signal sense ambiguity.

If a signal name appears suffixed by a minus sign (-), then that signal is active low. The minus sign is part of the signal name, and not an operator. As an example, the signal ACK- appears on several timing diagrams and the minus is there to remind the reader that a low on the timing diagram is the asserted state.

In state diagrams the asserted states appear as 1s. This makes the diagram easier to read than one with Ts and Fs. In any case there is no ambiguity because the boolean variables which are used in state diagrams are not circuit level signals. For example, the variable CDIT is used in a state diagram with a 1 being true, while the corresponding signal name in the schematic and the timing diagram is CDIT- with a low assertion level.

The backslash / is the inversion operator. This is similar to the BAR operator in boolean algebra, so /A has the same meaning as A. An operator does not signify activity level, so the inclusion of a signal suffix (- or blank) is additional information.

In PLD definitions and equations, the signal assertion level should only appear in the pin name declaration. PLD equations should then be written referring only to variable names as they appear in state diagrams and truth tables.

The design file for this CY7C330 application has not been included in this note, but is available from Cypress Semiconductor.

History

The SCSI standard evolved from the SASI controller specification by DTC and Shugart, which was a widely adopted parallel interface for disk controllers. The current SCSI standard is upwards compatible from this original specification.

Apart from the more rigorous timing and electrical specifications, most SCSI additions (i.e., reselection, arbitration, and synchronous mode) apply when the interface is being used as a network. If the sole use of the interface is to access a mass storage subsystem, then these features may be omitted and the resultant SCSI implementation will be smaller and faster.

The current SCSI interface is 8 bits wide, and it is possible to operate in asynchronous mode for a minimally featured interface at a rate of up to 16 megacycles. The interface may be widened to 16 bits at some time in the near future; if so, then the SCSI throughput rate will double to a theoretical maximum of about 32 megabytes per second.

The SCSI standard is likely to prevail in storage system interfaces. The only competing standard is ESDI which, being a serial data interface, has a much lower data throughput.

System Considerations

A block diagram of a minimal SCSI implementation is shown in *Figure 1.* Normally the mass store subsystem is inside the same enclosure as the computer; if it is not, then for emission considerations differential drivers and receivers should be used. In this application note, it is assumed that the flat cable SCSI bus is about a foot long so that transmission delays are minimal (5 ns).

The mass store subsystem consists of one or more disk drives or other high density storage devices, and one or more controllers with SCSI ports. Unused lines in the SCSI bus are not shown in *Figure 1.*

The computer system itself will access the SCSI controller from its own bus. For this example, a simple asynchronous interface has been implemented. This interface only one data strobe and there are two signals:

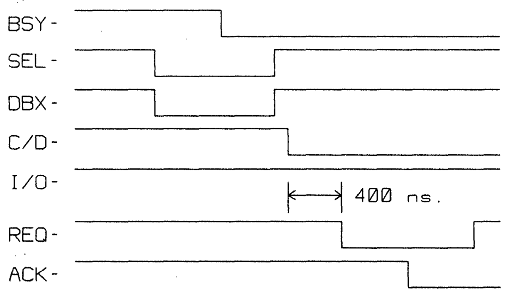


Figure 2. SCSI Command Phase Timing

RTS (Request To Send) and CTS (Clear To Send) to request or acknowledge data access cycles. These signals allow for the connection of a DMA device or another data interface.

The SCSI Transfer Protocol

A SCSI data access consists of a command transfer followed by a data transfer. The command transfer proceeds as follows:

1. The host waits for BSY to go inactive, then asserts SEL and one of the 8 data bits (to select one of 8 controllers).

2. The controller drives BSY active when this selection combination is detected.

3. The host releases SEL and the data bit used for selection.

4. The controller asserts C/D and REQ to read a command byte from the host.

5. The host outputs the first byte of the command and asserts ACK.

6. The controller accepts the data and deasserts REQ.

7. The host then deasserts ACK.

8. Steps 4 through 7 are repeated for 6 bytes (more in special cases).

Figure 2 illustrates this process.

After the command has been read in by the controller, the operation is either performed or aborted. After executing a command, a status byte (C/D asserted) is sent to the host to indicate success or an error condition.

If the command is a write command, then data is first transferred from the host to a buffer on the controller. After the data is written to the disk, a command complete status message is sent to the host.

If the command is a read command, then data is read from the disk, checked for validity, and passed to the host. Some controllers offer a "Fly-by" mode, which means data is passed to the host as soon as it is read and an error condition is signalled afterwards.

The normal data transfer protocol follows the above description (steps 4 to 7). At the end of the access, the status byte is transferred, then activity ceases. BSY goes inactive to signal the end of the access.

Interface Timing Considerations

There is one major delay and one minor delay to be observed during selection, and there is a data setup delay to be observed during data transfer.

For the host interface, under the single initiator option in the SCSI specification there is a 400 ns "bus settle delay" to be observed after BSY goes false, and before SEL is asserted. Additionally, SEL is to be deasserted at least two deskew delays after BSY is asserted. A deskew delay is 45 ns. Data is to be set up for a minimum of one deskew delay plus one cable skew delay (45ns + 10ns) before the ACK signal is given.

Like the host interface, the controller interface has timing constraints associated with selection and data access.

The controller implements the same data setup delay as the host, but the strobe which is accomodated from the controller side is **REQ**. The response to **SEL** must be shorter than 200 microseconds.

The setup time allowed for I/O and C/D [control signals] is specified as one "bus settle delay" or 400 ns. It is worth noting here that the response to SEL, and the various bus settle delay constraints are really system level response times, and need not be of concern in the hardware design at this level.

Performance Considerations

The CY7C330 is a Moore machine; there are no combinatorial paths from the inputs to the outputs. One consideration in state machine design with Moore machines is that the turnaround time or handshake delay to external signals can become the limiting factor in throughput. This problem is most obvious in asynchronous interfaces.

Figure 3 shows a hypothetical synchronized transfer cycle. This is the cycle as it could be implemented with a CY7C330 synchronous state machine, if the ACK signal was directly controlled by the CY7C330.

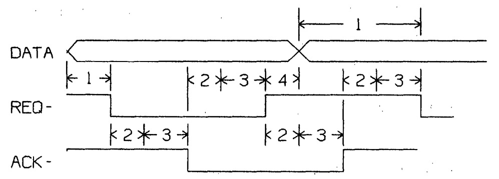


Figure 3. Synchronized Transfer Cycle

Definitions for Figure 3:

1. T_{SU}: 55 ns setup time for data
2. T_{LA}: Latency time delay; this consists of device propagation delays plus 0 or 1 clock cycles. For preliminary estimates, assume a 20 ns clock and 15 ns of delay.
3. T_C: Clock period.
4. T_D: Data delay (max) after REQ deasserted.

The time for one cycle using synchronized transfer cycles is about 180 ns. This cycle time corresponds to a throughput rate of just under 6 MHz, which is not as high a rate as the CY7C330 is capable of supporting.

The problem is that for every edge there is a synchronization or latency delay plus a clock delay before the corresponding handshake signal is given. These delays are undesireable and for the most part unnecessary, since the data path is capable of accepting data at a higher rate.

This result underscores the need for supervisory control over the handshake sequence. If the output data is ready and waiting, there is no need to delay the handshake sequence until the state machine synchronizes to the event and reacts. Likewise, if the input buffer is empty then it can be asynchronously loaded.

In the schematic (*Figure 10*) a NOR buffer is used to drive the output strobes, and to perform the asynchronous handshake, and to latch **ACK-** until the state machine has had sufficient time to react. The signal **CDIT-** is used by the CY7C330 to supervise the handshake sequence.

Transfer to the Controller

For transfers to the controller, the asynchronous signal that needs to be controlled is ACK- (active low acknowledge). This signal should go low soon after REQ- is asserted by the controller, but only after data has been setup for a minimum of 55ns. This signal should go high when REQ- is deasserted.

To guarantee that the state machine sees the cycle take place, **ACK-** is latched low until released by a controlling signal (**CDIT-**) that comes from the state machine. The same signal is used to hold off **ACK-** until the data setup has been met. (Refer to *Figure 10* for latch circuit details.)

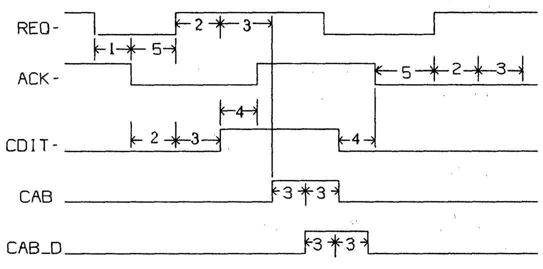


Figure 4. Host to Controller Transfer Timing

Another signal is required to clock data into the output register (CAB). This signal has a duration of two clock cycles for data setup timing. In *Figure 4* the signal CAB_D is a delayed feedback version of CAB which is used to add a delay cycle.

Definitions for Figure 4:

1. TAT: Asynchronous turnaround time (8 ns) X is the turnaround time in the other direction (8 ns)
2. TLA: Latency time delay; this consists of device propagation delays plus 0 or 1 clock cycles. For preliminary estimates, assume a 20 ns clock and 15 ns of delay. (25 ns average)
3. TC: Clock period. (20 ns)
4. TDO: Delay to output (15 ns)
5. Asynchronous turnaround time from controller end (8 ns)

Figure 4 shows the resultant transfer cycle to the controller from the host. The cycle time can be estimated from one REQ- rising edge to another. This time works out to an expected value of 108 ns.

The state diagram for the part of the controller that handles the interface timing is shown in *Figure 5*. At the start of the cycle, CDIT- is active because it is assumed that data has been at the interface for at least the setup requirement. CAB is the register clock for the output register, and it goes high after REQ- goes inactive (high) if there is data available (DAV, which is a logic function yet to be defined). The cycle then proceeds to completion and as CDIT- goes active, another cycle can start.

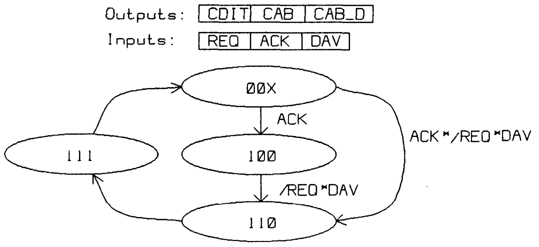


Figure 5. Controller to Host Transfer

The state diagram for the associated system transfer to the SCSI controller is shown in *Figure 6*. **E0-** and **E1-** are output enables for the two input registers; **CK0** and **CK1** are clocks for the same two registers; **CTS-** is a signal to the Host system that these registers are empty.

At the beginning [state 0000], **E0-** and **E1-** are inactive, the clocks are low, and **CTS-** is active [0]. When **DS-** is asserted, the clocks go high to capture the data, **E0-** goes active and **CTS-** goes inactive to signal that the registers have been loaded [state 1011, **CTS-** = 1].

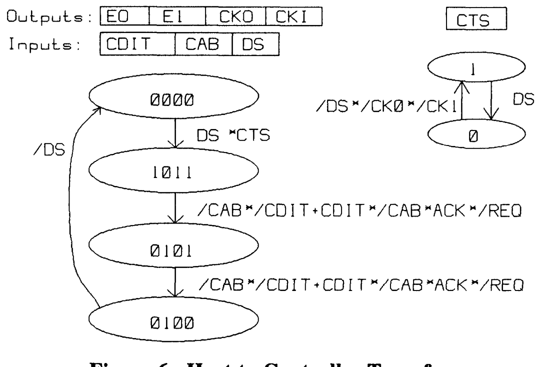


Figure 6. Host to Controller Transfer

When either CK0 or CK1 are high, data is considered available by the state machine in *Figure 5*, and consequently, DAV = CK0 + CK1. After CAB goes high, E1- goes active, E0- inactive, and CK0 goes low. [state 0101]

The next time CAB goes high, CK1 goes low to signify that the input registers are empty again. [state 0100] The state counter then automatically progresses [0000].

The machine waits for DS- to go inactive before allowing another cycle so that double clocking does not occur on one write cycle.

Transfer to the Host

When data is transferred to the Host from the controller, the handshake happens so quickly that there is a possibility that the interface will not see it, and for this reason ACK- must be latched until the CY7C330 signals [moves CDIT- high] to release it.

In this case, CDIT- is a signal that signifies that there is room in the receiving buffer for a data transfer. CBA is the clock for the input buffer and it goes high when CDIT- goes low or afterwards.

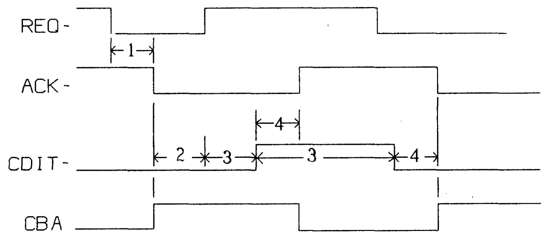


Figure 7. Controller to Host Transfer Timing

Definitions for *Figure 7*:

1. **TAT:** Asynchronous turnaround time (8 ns.) X is the turnaround time the other direction (8 ns.)
2. **TLA:** Latency time delay; this consists of device propagation delays plus 0 or 1 clock cycles. For preliminary estimates, assume a 20 ns. clock and 15 ns. of delay. (25 ns average)
3. **TC:** Clock period. (20 ns)
4. **TDO:** Delay to output (15 ns)

Figure 7 shows the relevant timing for this transfer cycle. The cycle time can be estimated from the rising edge of

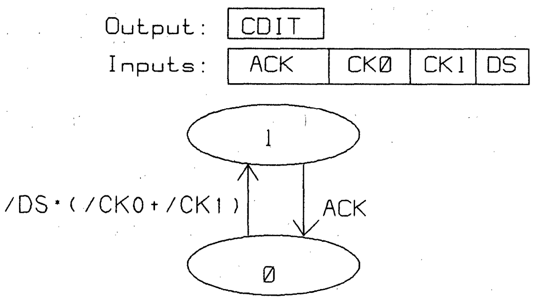


Figure 8. Controller to Host Transfer

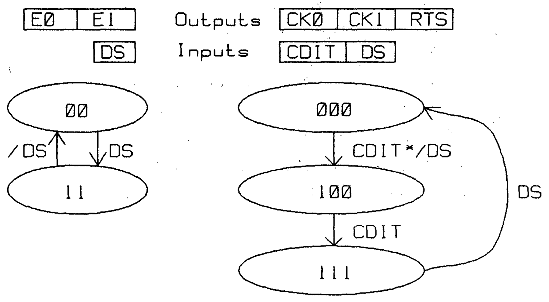


Figure 9. System to Host Adaptor Transfer

CDIT- to the next similar edge. In this case, it is reasonable to expect a cycle time of about 80 ns. *Figure 8* shows the state diagram for this cycle. *Figure 9* shows the state diagram for the system to interface transfer cycle.

Staging Considerations

Staging considerations include the initialization, startup, and change of direction of the interface. The signal I/O from the SCSI port mandates the direction of transfer, which changes during the process of command completion, so there is a need to make sure that the relevant state machines are all qualified by I/O.

A readback path is provided for the CPU on the Host system to be able to read the SCSI signals directly. The signal DS- is reserved for normal data, but the signals CS0- thru CS1- allow D0 on the system data bus to be used to read SCSI signals.

The following addresses apply:
CS0 = 0: enable readback to D0
CS0 = 1: disable readback
CS2,CS1: 00 - BSY
CS2,CS1: 01 - C/D
CS2,CS1: 10 - I/O
CS2,CS1: 11 - REQ

The reset function for SCSI Controllers is independent of the Host interface controller. In the schematic of *Figure 10*, the signal RST is set by the Host system and this simply forces the RST- signal low on the interface.

The controller can be reset at any time by asserting INIT- from the host system. If the code 001 is on CS2,CS1,CS0 then a select is performed: SEL- is pulled low until BSY- appears.

The transfer of data to the interface, in particular the device select code, should be done before the selection sequence is performed. After INIT- is released, data can be transferred normally and the REQ, ACK handshake will operate properly.

The transfer of diagnostic data (i.e., sense byte, errors) to the Host will be indicated by the DIAG- flag, which is set until INIT- is asserted.

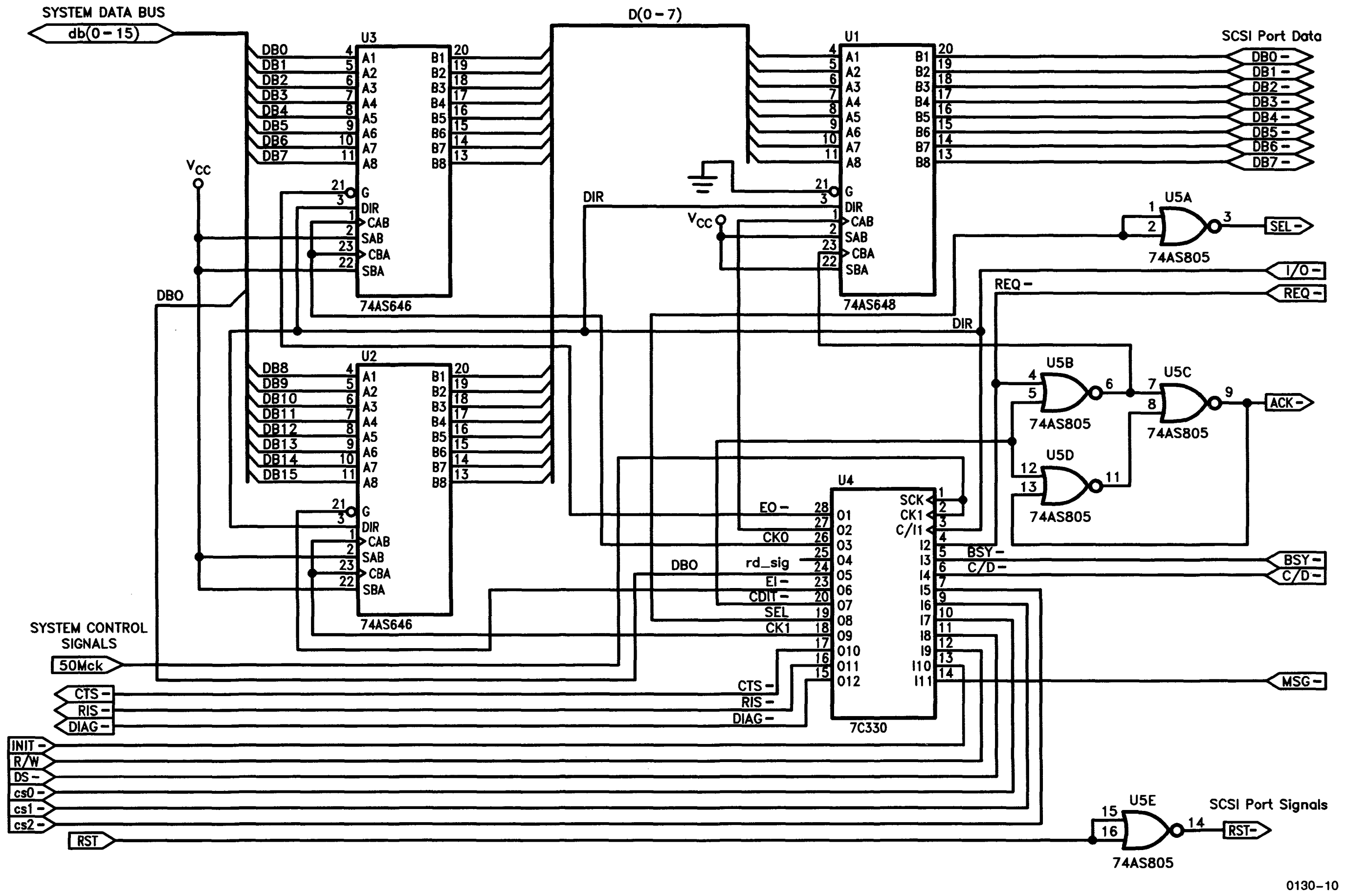


Figure 10. Host SCSI Port

NOTES:

CYPRESS SEMICONDUCTOR

Using the Cypress CY7C330 in Closed-Loop Servo Control

Introduction

This application note examines a common facet of engineering design--control systems--and offers an alternative to common implementations. An overview of control systems is discussed, along with several implementation strategies. The benefits and disadvantages of these implementations is briefly reviewed. Finally, the PLD-specific elements of a method are disclosed that use the Cypress CY7C330 as a key processing element that offloads the processing bandwidth requirements of a controlling CPU. This method has been successfully employed in a high-speed customer application--a laser mirror positioning servo.

Control Systems Overview - General Concepts

Control systems constitute a considerable portion of engineering design. Control system theory is applied to areas as diverse as pneumatic controls to economic models. There are numerous references available for an in-depth discussion of control theory. The mathematical analysis of their behavior relies heavily on a solid understanding of Laplace and z-transforms. Fortunately, we will limit ourselves to a practical discussion of control systems and briefly discuss the PID, (Proportional, Integral, Differential), method. Variations of the PID are used for approximately 80-90% of industrial control implementations.

Control systems are broadly divided into two major categories: open loop and closed loop. An open-loop system is one that generates outputs based on input conditions, but has no feedback from the output to verify or correct the output condition. Examples of open-loop systems include light switches (although one could reasonably argue that the human is the feedback loop), and self-timed, free-running traffic control signals. Closed-loop systems are those that provide information pertaining to the system status to the controller. Examples of closed-loop systems include the eye-brain system being used to read this line of text, the engine thermostat in most automobiles, and the print head of a dot-matrix printer. The closed-loop application we will discuss later consists of a mirror attached to a motor that can rotate 360 degrees in either direction. Closed-loop systems use information from the enviroment under control to influence the output. Control systems are typically represented with block diagrams as shown in *Figure 1.*

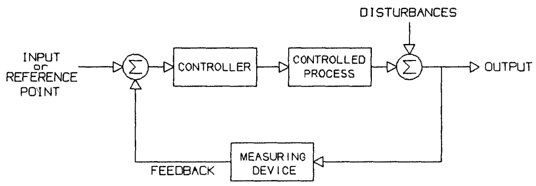


Figure 1. Closed-Loop Servo System

In a closed-loop design, numerous factors influence the system behavior. Among them are:

Input [I(t)]. The input to the system is the signal from an external source that is used to reference the steady-state behavior desired. In our mirror servo system, the steady-state output that we are attempting to attain is the absolute position of the mirror at a given location within a given percentage of accuracy. The input is also known as the reference or set point.

Summing Function. This is the section of the control system that determines the amount of error [E(t)] presently in the system. It is the difference between the input or reference point and the current state of the controlled environment. In a motor servo system, it is the difference between the target reference position desired and the current position of the motor. In an analog implementation, the summing function is usually implemented with an operational amplifier.

Controller. In most control systems there is a controller that has the error signal as an input and generates an output that attempts to reduce this error to within tolerable levels (ideally 0). The controller has an associated control mode that determines how the error signal is manipulated to produce a control signal. Common control modes include proportional, integral, differential, and the combination of these three --PID.

Controlled Device. The object of our effort is to have a controlled device perform satisfactorily. In our servo case, this is the motor.

Output [O(t)]. This is the physical entity to be controlled. In our automobile thermostat system, it is the temperature of the engine. In the servo postioned mirror example, it is the position of the mirror.

Disturbance [D(t)]. Any influence on the system that negatively affects the desired output is called a disturbance. In the automobile, operation in bumper-to-bumper traffic that reduces airflow through the radiator is a disturbance to the thermostat.

This is a partial list of the influences in a closed-loop control system, but those mentioned are the most significant for our example. A more complete discussion can be found in a good reference source.

Some of the parameters used to quanitize the behavior of control systems are listed below.

Accuracy. This is the difference between ideal and actual steady-state system behavior.

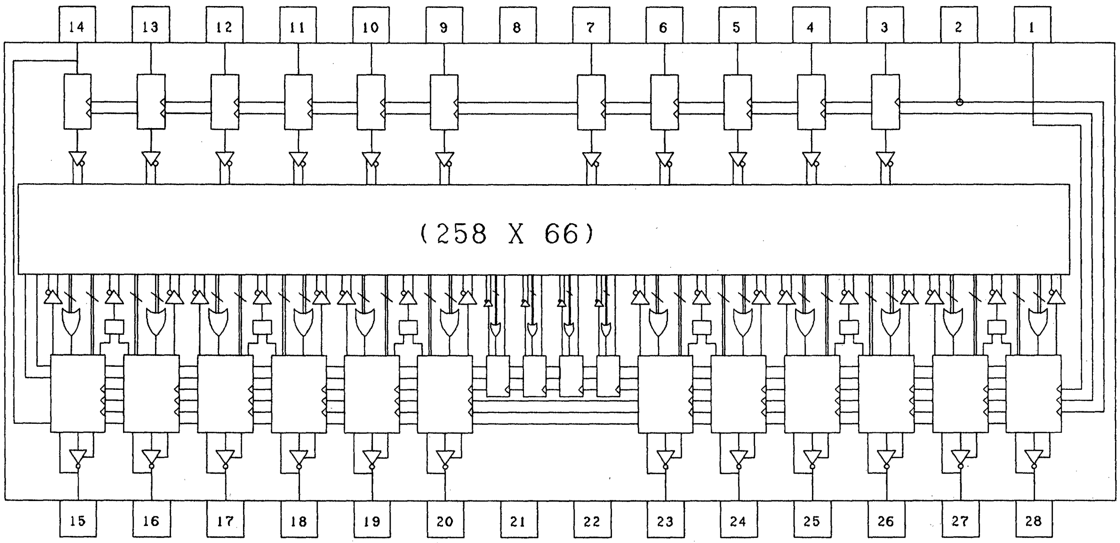


Figure 2. Cypress 7C330 PLD

Settling Time. This is the time required to reach steady state after the reference point is changed or set.

Percentage Overshoot. This is the difference between the reference point and the maximum excursion after passing through the reference point.

Jitter. This is a condition that occurs when the controlling element improperly overcompensates for an overshoot of the reference point. This results in an undershoot that is again overcompensated for and produces overshoot. This can result in an unstable oscillatory condition where the reference point is never obtained, or it can increase the settling time.

Rise Time. This is the time required for the system's output to increase from 10% to 90% of the final value.

Control Systems Overview - Implementations

Control system implementations vary from purely analog to completely digital. Many popular implementations use a hybrid of digital and analog techniques. The approach we will examine uses a digital element to perform the summing, control, and part of the feedback section. This approach and the pure analog method are possibly the most often used. Naturally there are tradeoffs in each approach. Analog systems continuously perform the summing function (usually with an op-amp) and therefore are usually more stable because they are immmune to the problems associated with the quantitization of data. Digital hybrids offer improved sensitivity, greater immunity to noise, better resolution, minimized drift, more flexibility, and are usually easier to design at a lower cost.

With the hybrid approach, several methods are used for the controller. With the advent of the microprocessor, it is relatively easy to implement the controller and the summing function on chip. When this approach is taken, a number of algorithms can be used to generate the control signal. The simplest is proportional control. In proportional control, the correction made is proportional to the error signal. The value that the error is scaled by is the proportionality constant or gain. Proportional control offers an intuitively reasonable solution: the larger the error, the larger the corrective signal. Using integral control, the corrective signal is based on the time integral of the error multiplied by a weighting factor. This value is typically calculated using a numeric approximation. Integral control is usually combined with proportional control to increase the accuracy or reduce the steady-state error. Finally, the corrective signal with derivative control is the derivative of the error signal over time multipled by a weighting factor. Again, a numeric approximation is used to calcluate the derivative. This addition to proportional control contributes a stabilizing influence to the system. However, it is often omitted in "noisy" systems due to its effect of amplifying high-frequency disturbances. When combined, these three methods constitute proportional + integral + derivative, or PID control. The influences of the integral and derivative methods on PID can be verified with analysis based on Laplace transforms. The cost of using PID is in the reduction in the available bandwidth of the processor to perform other tasks. Also, a finite amount of time is required to calculate the output value.

Another factor to consider in a hybrid control system is the sampling/processing rate of the system. Several references indicate that the sampling rate for a closed-loop control system should be significantly above the minimum dictated by Shannon's sampling theorem. Rather than being able to operate at the Nyquist frequency of twice the highest frequency sampled, it is often recommended that a sampling rate of eight to ten times the highest sampled frequency be used. The reasons cited include an uncertainty associated with determing the highest frequency component of the sampled signal, and the possibility of aliasing or a decrease in system stability occurring due to the selection of too low a sampling rate. Again, in a microprocessor-based implementation, the available processor bandwidth is quickly consumed as the sampling rate is increased to maximize stability.

Using the Cypress CY7C330 in Servo Control

In response to an ever-increasing system workload, the Cypress CY7C330 has been utilized in a high-speed servo control system to offload the microprocessor. This particular application positioned a mirror to form images with a laser beam. The previous implementation used a 68000 microprocessor in the servo loop, as detailed above. As the number of tasks on the 68000 increased, the processors ability to maintain a stable servo system became marginal. The design engineer's goal was to meet the servo loop stability requirements and the additional processor system throughput needs with a minimum of additional cost and complexity. The solution offered here is both simple and efficient.

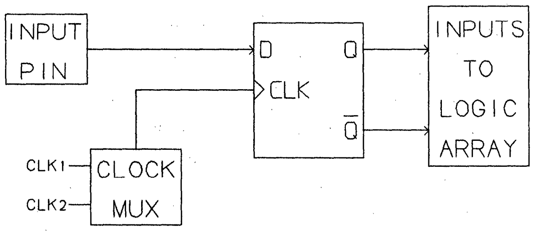


Figure 3. 7C330 Dedicated Input Register

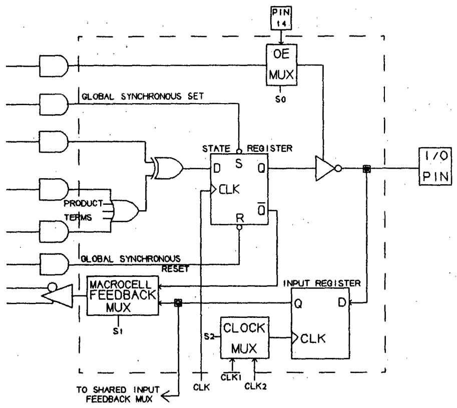


Figure 4. 7C330 I/O Macrocell

Several features of the 7C330 (*Figure 2*) are fundamental to understanding this design. The first is the dedicated input registers (*Figure 3*). These registers allow data to be loaded onto the chip with either of two data input clocks, CLK1 (pin 2) or CLK2 (pin 3). The choice of input clock is done at program time via an EPROM configuration fuse. The I/O macrocells (*Figure 4*) also features input registers, again with two clocks for data entry. In some applications, such as up/down counters, the ability to three-state the macrocell output drivers and load data into the macrocell input register allows the designer to use these macrocell input registers to hold reference values (such as counter upper/lower limit). In our design, the macrocell input registers are used to store the calculated target position of the mirror, and are clocked in with clk2. The dedicated input registers in this design are used while actively controlling the servo for loading the present mirror position from the servo loop. When in command mode, the dedicated input registers are loaded with data from the microprocessor that is used to calculate a new target position. In either case, the dedicated input registers are loaded with clk1. Subsequent diagrams will show the the 7C330's dedicated input registers and macrocells in a simplified format that highlights our servo implementation.

Let's take a look at the fundamentals of the design. Referring to *Figure 1*, we see that the basic mechanism of control loops is proportional feedback of the error signal. If we were to design this loop as a self-contained co-processor to the main CPU, the CPU would only be required to input the reference point to which we want to move the mirror. Now the CPU would no longer be required to perform the control algorithm at a pace equal to the sampling rate. Essentially, the processor could "set and forget" the servo co-processor. One way to implement this servo co-processor would be through the addition of an additional microprocessor. This would add additional hardware (CPU, RAM, ROM, Clock, I/O, Interrupt Control, etc.), additional software, and possibly require an In-Circuit Emulator for development if a low-cost microcontroller were used. We might use an analog servo controller, but the accuracy requirements preclude this when drift is considered. Instead we used several simple PLDs in a hybrid control-loop implementation.

The system block diagram in *Figure 5* shows the *general* approach used. Three CY7C330s are used that each generate an 8-bit accumulate for 24 bits of precision. The microprocessor provides to the CY7C330s a 24-bit position reference target for the mirror. This 24-bit value is latched into the 7C330s on-board registers. The 330 performs the summing and proportional feedback function of the control loop. The 24-bit desired position is compared to the present position that is maintained

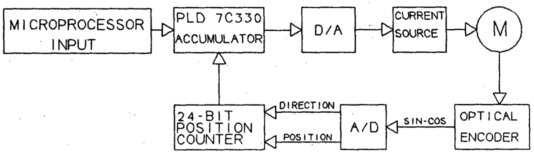
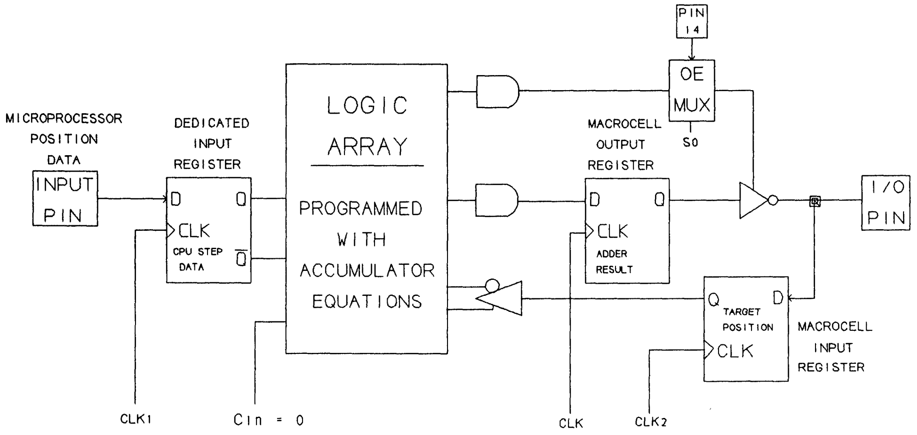


Figure 5. 7C330 Servo Control Loop

in an external 24-bit current position counter. The result is the error multiplied by a fixed unity gain. This proportional control signal is then converted to an analog signal that controls the positioning mirror's motor after being converted to a current level. The shaft of the motor has an optical encoder that creates a sin-cos analog signal. When converted to a digital signal, this gives a direction of rotation indication and a pulse that increments or decrements an external 24-bit present-position counter. This allows the loop to operate as fast as the slowest of the following elements: the 7C330s configured as a multistage accumulator/subtractor, the D/A conversion time, or the A/D conversion time. The host microprocessor is completely decoupled from the servo loop. Should the microprocessor halt, the servo circuitry will continue to maintain the desired reference position without intervention.

Of course the actual implementation is slightly more complex than the block diagram indicates. Essentially, the CY7C330 macrocell output registers are programmed to act as an accumulator. This accumulator generates a value that is one of two things depending on the mode of operation--either a new target position of the servo motor or the proportional error feedback value to the servo. When the system is started, the macrocell input registers wake up with an initial value of 0. These macrocell input registers are dedicated to holding the current target position of the motor. At the same time the external position counter is also set to zero. Then the microprocessor steps the target position until an alignment sensor is targeted by the laser.

This is accomplished using the following steps. First the outputs of the external 24-bit position counter are placed into a three-state condition. These outputs are shared with the outputs of the microprocessor as inputs to the dedicated input registers. The processor drives a step value onto the inputs, which is clocked into the 7C330's dedicated input registers with the CLK1 pin. Then, this value is added (via the PLD equations described later) to the current value in the macrocell input registers on the rising edge of the CLK pin. The result of this addition is now in the macrocell output registers and is clocked with CLK2 into the same macrocell input registers that were a source value for the add. Thus the 7C330s in this mode use the current value



Target Update Mode Operation Seqence

(1) With external position counter's output three-state, host microprocessor drives position step data.
(2) Step data (provided in 2's complement form if a subtract is desired) is loaded into the 330 with CLK1.
(3) Step data is added or subtracted from present target position with logic equations to create new target position.
(4) New target position is clocked into macrocell output registers with CLK.
(5) On CLK2, the new target position is clocked into the macrocell input register.

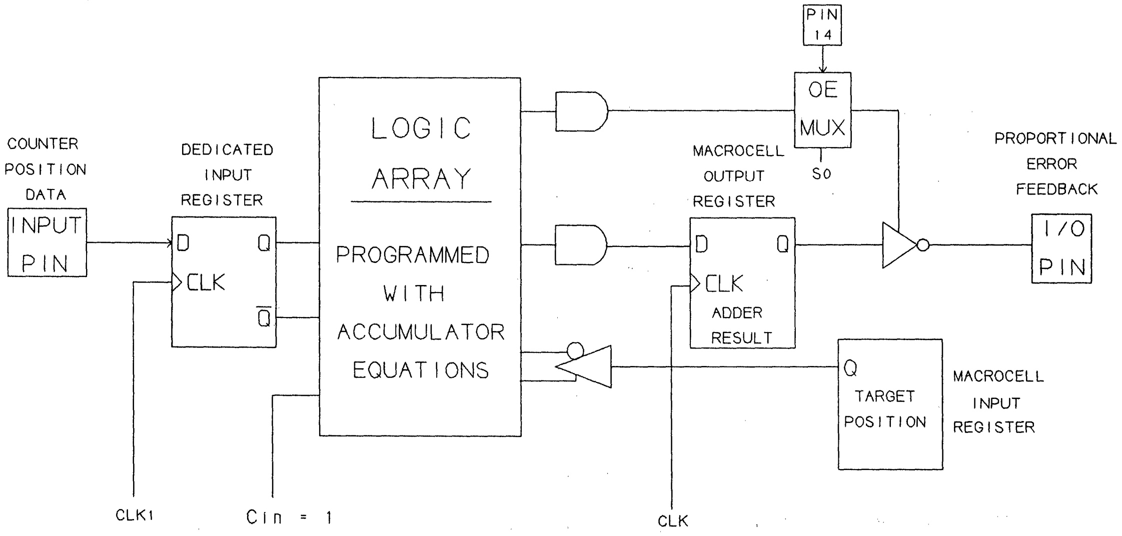
Figure 6. Target Update Mode

on the dedicated input pins to adjust the target postion in the macrocell input registers with an accumulate cycle. This target position update cycle is pictured in *Figure 6.* Data from the microprocessor is always provided as a delta or step from the current position. The accumulate can be either an add or subtract. Subtracts are accomplished by providing the step data from the microprocessor in 2's-complement form. After alignment, the position and accumulator values are reset to zero and the system is ready for operation.

In operation, the outputs from the microprocessor are three-state and the value from the 24-bit position counter is loaded into the the dedicated input registers. This value is always provided in a 2's-complement form by inverting the outputs of the position counter (1's complement) and setting carry in to one. This value is thus subtracted from the present target position value stored in the macrocell input registers to form the proportional error feedback value that is used to control the servo motor. This servo control mode is shown in *Figure 7.*

Again, the actual implementation details are different from the conceptual block diagram. The digital-to-analog converter does not need a 24-bit digital value for control. In practice, an 8-bit D/A value is used that is biased such that the 8th bit provides direction control (clockwise vs. counterclockwise). In the actual design, the upper 16 bits from the two most significant 330s are tested for rail high and low conditions and generate two offscale bits each for these conditions. The seven low-order bits, along with the four offscale bits, are passed to a second PLD (22V10) that drives the output to the D/A in the proper direction (eighth bit), with the proper magnitude. If the four offscale bits indicate that the upper bits are all close to 0, the seven bits to the D/A are masked to 0. Likewise if the upper bits are mostly 1, the D/A bits are set to 1. The determination of how to use the offscale bits for compensation in the second PLD is specific to a given application.

The backbone of the logic required to create this design is the implementation of an accumulator with the CY7C330. The logic required for implementing a synchronous full adder is described by an equation for the sum and an equation for the carry of a given bit.



Control Mode Operation Sequence

(1) External 24-bit position data (in 1's complement form) is loaded into the 7C330's dedicated input register with CLK1.
(2) With carry in set to 1, logic equations subtract current position from target position to form error amount.
(3) Error result is clocked into macrocell output register with CLK and is available to servo motor interface.

Figure 7. Control Mode

The equation for the sum, S at bit position n with inputs A, B, and carry in, Cin, is:

Sn = (An XOR Bn XOR CIN).

The equation for the carry out is:

COUTn = (An * Bn) + (An * Cn-1) + (Bn * CN-1)

The equations for a 4-bit synchronous adder requiring

```
/* Four Bit Adder - General Case */

Inputs:  An, Bn ; Inputs to be added at Bit n
         CIN    ; Carry in to Adder

Outputs: Sn     ; Sum out for Bit n
         Cn     ; Carry out from adder stage n

/* Equations to be reduced */

S0 = A0 XOR B0 XOR CIN
C0= (A0 * B0) + (A0 * CIN) + (B0 * CIN)

S1 = A1 XOR B1 XOR C0
C1 = (A1 * B1) + (A1 * C0) + (B1 * C0)

S2 = A2 XOR B2 XOR C1
C2 = (A2 * B2) + (A2 * C1) + (B2 * C1)

S3 = A3 XOR B3 XOR C2
C3 = (A3 * B3) + (A3 * C2) + (B3 * C2)

/* C3 = = Carry Out of Four Bit Adder */
```

Figure 8. Equations for Four Bit Adder

four clocks to complete are shown in Figure 8. Since the objective is to calculate a complete 24-bit sum as quickly as possible, the equation for carry out (C0) from the first bit of the adder can be substituted into the equation for the second bit of the adder. This allows the first two bits to be added in a single clock cycle. Likewise, the equation for the carry out from the second bit can be substituted into the equation for the third sum, and so on. This resulting equations for three bits of substitution are shown in *Figure 9.* The 7C330's XOR product term is useful in reducing the number of product terms required for a given sum bit. However, even after boolean reduction with utilization of the 7C330's XOR product term, the fourth bit of the adder requires 30 product terms for the sum bit and 31 product terms for the carry out bit to generate a 4-bit result in a single

```
/* Synchronous 3 bit adder - derivative of General Case */
/* Uses substitution of Carry Out in first 3 bits to generate 3 bit
result in one clock cycle */

S0 = A0 XOR B0 XOR CIN
/ * C0= (A0 * B0) + (A0 * CIN) + (B0 * CIN) */

S1 = A1 XOR B1 XOR [(A0 * B0) + (A0 * CIN) + (B0 *
CIN)]

/* C1 = (A1 * B1)
 + (A1 * [(A0 * B0) + (A0 * CIN) + (B0 * CIN)])
 + (B1 * [(A0 * B0) + (A0 * CIN) + (B0 * CIN)]) */

S2 = A2 XOR B2 XOR
{(A1 * B1)
 + (A1 * [(A0 * B0) + (A0 * CIN) + (B0 * CIN)])
 + (B1 * [(A0 * B0) + (A0 * CIN) + (B0 * CIN)])}

C2 = (A2 * B2)
 + (A2 *
{(A1 * B1)
 + (A1 * [(A0 * B0) + (A0 * CIN) + (B0 * CIN)])
 + (B1 * [(A0 * B0) + (A0 * CIN) + (B0 * CIN)])})
 + (B2 *
{(A1 * B1)
 + (A1 * [(A0 * B0) + (A0 * CIN) + (B0 * CIN)])
```

Figure 9. Equations for a Synchronous 3-Bit Adder

clock cycle. Since the maximum number of product terms for a given macrocell in the 7C330 is 19, the accumulate process must be accomplished over multiple 3-bit stages. The addition of the first three bits will be complete after one clock cycle, the second three bits after two cycles, and so on. Therefore the complete 24-bit accumulate requires nine clock cycles implemented on three 7C330s. With 66 MHz devices this translates to a complete calculation cycle in 120 nanoseconds.

The minimized equations for one of the three 8-bit adder stages is shown in Appendix A. The syntax used in this example is the Cypress PLD ToolKit. Variables B0 - B7 are the eight dedicated inputs that are sourced from either the microprocessor or the 24-bit position counter. INCLK is the clk1 pin on the 7C330 that is used to clock in the B0-B7 variables. CIN is the Carry in from external logic (set to one for subtraction when in control mode on the first 8-bit adder stage) or from the previous stage of the adder. A0 - A7 are the sum outputs for either target update or control mode. If the processor is updating the target position by a step increment, A0 - A7 are loaded into the macrocell input registers with clk2 (named ACLK). When this new position update is being loaded, the output drivers of the macrocells are not three-stated with the OE pin or a product term equation. This allows the macrocell output registers (which have the newly calculated target position) to be loaded into the macrocell input registers (which are used to hold the target position) with ACLK. C2 and C5 are internal carry-out bits generated from the first and second 3-bit adder stages respectively. Finally, COUT is the carry out generated as either the final carry out or as the input to carry in of the next 8-bit adder stage.

Appendix B shows the implementation of the two upper 7C330 stages. The equations for the accumulator function are the same as in the previous equation. The additions here are the equations for detecting rail conditions and generating the offscale bits. These bits are generated to minimize the number of inputs required for the subsequent PLD that feeds the D/A converter. The use of these bits is dependent on the application.

Conclusion

This application note discusses some of the basic concepts involved in control theory and several implementation strategies. Control theory is a very wide subject area, and the underlying mathematical analysis of these systems is beyond the scope of this paper. Several good sources for further reading are listed below. An example of a PLD servo loop co-processor that utilizes proportional control with unity gain has been detailed. This example is intended to cover the specifics involved in utilizing the Cypress PLD 7C330 in this type of application. There are numerous other hardware implementation details that are left to the designer (such as D/A design, feedback design, lead/lag compensation, etc.). Our intent is to focus on a different approach to implementing a closed-loop servo controller with the Cypress 7C330 as the central element, and to disclose the details unique to the 7C330. Implementation of the 24-bit Up/Down position counter with the 7C330 is discussed in the "66 MHz 7C330 Synchronous State Machine" application note.

References

Digital Control Systems - Theory, Hardware, Software.
- Houpis & Lamont; 1985; McGraw-Hill; N.Y.

Engineering Applications of Microcomputers
- Ball & Pratt; 1986; Prentice Hall Int'l (UK) Ltd; N.J.

Digital Control Systems
- Kuo; 1980; Holt, Rinehart, & Winston, Inc.; N.Y.

Analog and Digital Control Systems
- Gayakwad & Sokoloff; 1988; Prentice Hall; N.J.

Computer Control of Machines & Processes
- Bollinger & Duffie; 1988; Addison - Wesley; N.Y.

Acknowledgement

The author would like to extend his thanks to Delmar Curtis. His experience in the design of the circuit on which this application note is based, and willingness to share the basic principals involved proved invaluable.

Appendix A. PLD ToolKit Code for an 8-Bit Accumulator

```
        {Cypress Semiconductor - 8-bit accumulator  - June 14, 1989}

CY7C330;

CONFIGURE;                    { Dedicated input registers. Default configuration is use of pin 2 for clock }

Outclk(node=1),
Inclk(node=2),
Aclk(node=3),
CIN(node=4),
B0(node=5),
B1(node=6),
B2(node=7),
B3(node=9),
B4(node=10),
B5(node=11),
B6(node=12),
B7(node=13),
oe(node=14),

{Output nodes assigned to maximize available product term utilization. In the following declarations, the 7C330's
macrocell outputs are configured as follows:
        ireg--This sets the macrocell feedback MUX for feedback from the macrocell input register instead of the
              (default) macrocell output register (rgd)

        iclk=3--This selects the clock on pin 3 instead of the default (used for the inputs above) of clock on pin 2 for the
                macrocell input register

        IOP--Same as ireg.

        nenbpt--Selects OE control from pin 14 instead of a product term }

A0(node=28,iop,iclk=3,ireg,nenbpt),        { Sum 0 / Accum. Feedback Register 0 }
A1(node=15,iop,iclk=3,ireg,nenbpt),        { Sum 1 / Accum. Feedback Register 1 }
A2(node=20,iop,iclk=3,ireg,nenbpt),        { Sum 2 / Accum. Feedback Register 2 }
A3(node=17,iop,iclk=3,ireg,nenbpt),        { Sum 3 / Accum. Feedback Register 3 }
A4(node=26,IOP,iclk=3,ireg,nenbpt),        { Sum 4 / Accum. Feedback Register 4 }
A5(node=23,IOP,iclk=3,ireg,nenbpt),        { Sum 5 / Accum. Feedback Register 5 }
A6(node=19,IOP,iclk=3,ireg,nenbpt),        { Sum 6 / Accum. Feedback Register 6 }
A7(node=24,IOP,iclk=3,ireg,nenbpt),        { Sum 7 / Accum. Feedback Register 7 }
COUT(node=18,nenbpt),                      { Carry out  }
C2(node=32),                               { Carry 2 - Hidden }
C5(node=34),                               { Carry 5 - Hidden }

        { Available nodes    -# P.T.'s  }
        { I/O macrocell     - 16 - 19   }
        { I/O macrocell     - 25 - 17   }
        { I/O macrocell     - 27 - 19   }
        { hidden macrocell - 31 - 13    }
        { hidden macrocell - 33 - 11    }

        {End of configuration section}
```

Appendix A. PLD ToolKit Code for an 8-Bit Accumulator (continued)

```
        {Logic equation section}

EQUATIONS;

        {A0: 2 product terms, pin 28: 9 P.T. Available}

/A0 =   <XSUM> CIN
        <SUM> /A0 * /B0
            +    A0 * B0;

        {A1: 6 product terms, pin 15: 9 P.T. Available}

/A1 =   <XSUM> /A1
        <SUM> B1 * /B0 * /CIN
            +    /B1 * B0 * CIN
            +    /B1 * A0 * CIN
            +    /B1 * A0 * B0
            +     B1 * /A0 * /CIN
            +     B1 * /A0 * B0;

        {A2: 14 product terms, pin 20: 15 P.T. Available}

/A2 =   <XSUM> /A2
        <SUM>  B2 * /A1 * /B1
            +    /B2 * B1 * B0 * CIN
            +    /B2 * A1 * B0 * CIN
            +    /B2 * B1 * A0 * CIN
            +    /B2 * A1 * A0 * CIN
            +    /B2 * B1 * A0 * B0
            +    /B2 * A1 * A0 * B0
            +    B2 * /B1 * /B0 * /CIN
            +    B2 * /A1 * /B0 * /CIN
            +    /B2 * A1 * B1
            +    B2 * /B1 * /A0 * /CIN
            +    B2 * /A1 * /A0 * /CIN
            +    B2 * /B1 * /A0 * /B0
            +    B2 * /A1 * /A0 * /B0;

        {C2: 15 product terms, virtual pin 32: 17 P.T. Available}

C2 =    <SUM>  B2 * B1 * B0 * CIN
            +    A2 * B1 * B0 * CIN
            +    B2 * A1 * B0 * CIN
            +    A2 * A1 * B0 * CIN
            +    B2 * B1 * A0 * CIN
            +    A2 * B1 * A0 * CIN
            +    B2 * A1 * A0 * CIN
            +    A2 * A1 * A0 * CIN
            +    B2 * B1 * A0 * B0
            +    A2 * B1 * A0 * B0
            +    B2 * A1 * A0 * B0
            +    A2 * A1 * A0 * B0
            +    B2 * A1 * B1
            +    A2 * A1 * B1
            +    A2 * B2;
```

Appendix A. PLD ToolKit Code for an 8-Bit Accumulator (continued)

```
        {A3: 2 product terms, pin 17: 11 P.T. Available}

/A3 =  <XSUM> C2
      <SUM> /A3 * /B3
        +      A3 *  B3;

        {A4: 6 product terms, pin 26: 11 P.T. Available}

/A4 =  <XSUM> /A4
      <SUM>  B4 * /B3 * /C2
        +     /B4 *  B3 *  C2
        +     /B4 *  A3 *  C2
        +     /B4 *  A3 *  B3
        +      B4 * /A3 * /C2
        +      B4 * /A3 *  B3;

        {A5: 14 product terms, pin 23: 15 P.T. Available}

/A5 =  <XSUM> /A5
      <SUM>  B5 * /A4 * /B4
        +     /B5 *  B4 *  B3 *  C2
        +     /B5 *  A4 *  B3 *  C2
        +     /B5 *  B4 *  A3 *  C2
        +     /B5 *  A4 *  A3 *  C2
        +     /B5 *  B4 *  A3 *  B3
        +     /B5 *  A4 *  A3 *  B3
        +      B5 * /B4 * /B3 * /C2
        +      B5 * /A4 * /B3 * /C2
        +     /B5 *  A4 *  B4
        +      B5 * /B4 * /A3 * /C2
        +      B5 * /A4 * /A3 * /C2
        +      B5 * /B4 * /A3 * /B3
        +      B5 * /A4 * /A3 * /B3;

        {C5: 15 product terms, virtual pin 34: 19 P.T. Available}

C5 =   <SUM> B5 * B4 * B3 * C2
        +     A5 * B4 * B3 * C2
        +     B5 * A4 * B3 * C2
        +     A5 * A4 * B3 * C2
        +     B5 * B4 * A3 * C2
        +     A5 * B4 * A3 * C2
        +     B5 * A4 * A3 * C2
        +     A5 * A4 * A3 * C2
        +     B5 * B4 * A3 * B3
        +     A5 * B4 * A3 * B3
        +     B5 * A4 * A3 * B3
        +     A5 * A4 * A3 * B3
        +     B5 * A4 * B4
        +     A5 * A4 * B4
        +     A5 * B5;
```

Appendix A. PLD ToolKit Code for an 8-Bit Accumulator (continued)

```
{A6:  2 product terms, pin 19: 13 P.T. Available}


/A6 =  <XSUM>  C5
       <SUM>  /A6 * /B6
         +      A6 *  B6;

       {A7:  6 product terms, pin 24: 13 P.T. Available}

/A7 =  <XSUM> /A7
       <SUM>   B7 * /B6 * /C5
         +    /B7 *  B6 *  C5
         +    /B7 *  A6 *  C5
         +    /B7 *  A6 *  B6
         +     B7 * /A6 * /C5
         +     B7 * /A6 *  B6;

       {COUT:  7 product terms, pin 18: 17 P.T. Available}

/COUT  = <SUM>  /B7 * /B6 * /C5
         +    /A7 * /B6 * /C5
         +    /B7 * /A6 * /C5
         +    /A7 * /A6 * /C5
         +    /B7 * /A6 * /B6
         +    /A7 * /A6 * /B6
         +    /A7 * /B7;

       {End of file.}
```

Appendix B. PLD ToolKit Code for an Accumulator with Rail Condition

```
        {Mark Aaldering - Cypress Semiconductor - 8-bit accumulator with rail condition outputs - June 14, 1989}

CY7C330;

CONFIGURE;                          { Dedicated input registers. Default configuration is use of pin 2 for clock }

Outclk(node=1),
Inclk(node=2),
Aclk(node=3),
CIN(node=4),
B0(node=5),
B1(node=6),
B2(node=7),
B3(node=9),
B4(node=10),
B5(node=11),
B6(node=12),
B7(node=13),
oe(node=14),

{Output nodes assigned to maximize available product term utilization. In the following declarations, the 330's
macrocell outputs are configured as follows:

        ireg--This sets the macrocell feedback MUX for feedback from the macrocell input register instead of the
              (default) macrocell output register (rgd)

        iclk=3--This selects the clock on pin 3 instead of the default (used for the inputs above) of clock on pin 2 for the
                macrocell input register

        IOP--Same as ireg.

        nenbpt--Selects OE control from pin 14 instead of a product term }

A0(node=28,iop,iclk=3,ireg,nenbpt),     { Sum 0 / Accum. Feedback Register 0 }
A1(node=15,iop,iclk=3,ireg,nenbpt),     { Sum 1 / Accum. Feedback Register 1 }
A2(node=20,iop,iclk=3,ireg,nenbpt),     { Sum 2 / Accum. Feedback Register 2 }
A3(node=17,iop,iclk=3,ireg,nenbpt),     { Sum 3 / Accum. Feedback Register 3 }
A4(node=26,iop,iclk=3,ireg,nenbpt),     { Sum 4 / Accum. Feedback Register 4 }
A5(node=23,iop,iclk=3,ireg,nenbpt),     { Sum 5 / Accum. Feedback Register 5 }
A6(node=19,iop,iclk=3,ireg,nenbpt),     { Sum 6 / Accum. Feedback Register 6 }
A7(node=24,iop,iclk=3,ireg,nenbpt),     { Sum 7 / Accum. Feedback Register 7 }
COUT(node=18,nenbpt),                   { Carry Out }
C2(node=32),                                    { Carry 2 - Hidden }
C5(node=34),                                    { Carry 5 - Hidden }
R0(node=16,nenbpt),                     { Rail Bit 0 }
R1(node=25,nenbpt),                     { Rail bit 1 }

        { Available nodes      #   P.T.'s }
        { I/O macrocell      - 27 - 19    }
        { Hidden macrocell - 31 - 13      }
        { Hidden macrocell - 33 - 11      }

        {End of configuration section}
```

Appendix B. PLD ToolKit Code for an Accumulator with Rail Condition (continued)

```
{Logic equation section}
EQUATIONS;

        {A0: 2 product terms, pin 28: 9 P.T. Available}

/A0 =   <XSUM> CIN
        <SUM> /A0 * /B0
          +     A0 *  B0;

        {A1: 6 product terms, pin 15: 9 P.T. Available}

/A1 =   <XSUM>/A1
        <SUM>  B1 * /B0 * /CIN
          +   /B1 *  B0 *  CIN
          +   /B1 *  A0 *  CIN
          +   /B1 *  A0 *  B0
          +    B1 * /A0 * /CIN
          +    B1 * /A0 *  B0;

        {A2: 14 product terms, pin 20: 15 P.T. Available}

/A2 = <XSUM> /A2
      <SUM>  B2 * /A1 * /B1
        +   /B2 *  B1 *  B0 *  CIN
        +   /B2 *  A1 *  B0 *  CIN
        +   /B2 *  B1 *  A0 *  CIN
        +   /B2 *  A1 *  A0 *  CIN
        +   /B2 *  B1 *  A0 *  B0
        +   /B2 *  A1 *  A0 *  B0
        +    B2 * /B1 * /B0 * /CIN
        +    B2 * /A1 * /B0 * /CIN
        +   /B2 *  A1 *  B1
        +    B2 * /B1 * /A0 * /CIN
        +    B2 * /A1 * /A0 * /CIN
        +    B2 * /B1 * /A0 * /B0
        +    B2 * /A1 * /A0 * /B0;

        {C2: 15 product terms, virtual pin 32: 17 P.T. Available}

C2 = <SUM>  B2 * B1 * B0 * CIN
       +    A2 * B1 * B0 * CIN
       +    B2 * A1 * B0 * CIN
       +    A2 * A1 * B0 * CIN
       +    B2 * B1 * A0 * CIN
       +    A2 * B1 * A0 * CIN
       +    B2 * A1 * A0 * CIN
       +    A2 * A1 * A0 * CIN
       +    B2 * B1 * A0 * B0
       +    A2 * B1 * A0 * B0
       +    B2 * A1 * A0 * B0
       +    A2 * A1 * A0 * B0
       +    B2 * A1 * B1
       +    A2 * A1 * B1
       +    A2 * B2;
```

Appendix B. PLD ToolKit Code for an Accumulator with Rail Condition (continued)

```
        {A3: 2 product terms, pin 17: 11 P.T. Available}

/A3 = <XSUM> C2
      <SUM> /A3 * /B3
        +     A3 *  B3;

        {A4: 6 product terms, pin 26: 11 P.T. Available}

/A4 = <XSUM> /A4
      <SUM>  B4 * /B3 * /C2
        +   /B4 *  B3 *  C2
        +   /B4 *  A3 *  C2
        +   /B4 *  A3 *  B3
        +    B4 * /A3 * /C2
        +    B4 * /A3 *  B3;

        {A5: 14 product terms, pin 23: 15 P.T. Available}

/A5 = <XSUM> /A5
      <SUM>  B5 * /A4 * /B4
        +   /B5 *  B4 *  B3 *  C2
        +   /B5 *  A4 *  B3 *  C2
        +   /B5 *  B4 *  A3 *  C2
        +   /B5 *  A4 *  A3 *  C2
        +   /B5 *  B4 *  A3 *  B3
        +   /B5 *  A4 *  A3 *  B3
        +    B5 * /B4 * /B3 * /C2
        +    B5 * /A4 * /B3 * /C2
        +   /B5 *  A4 *  B4
        +    B5 * /B4 * /A3 * /C2
        +    B5 * /A4 * /A3 * /C2
        +    B5 * /B4 * /A3 * /B3
        +    B5 * /A4 * /A3 * /B3;

        {C5: 15 product terms, virtual pin 34: 19 P.T. Available}

C5 = <SUM> B5 * B4 * B3 * C2
        +   A5 * B4 * B3 * C2
        +   B5 * A4 * B3 * C2
        +   A5 * A4 * B3 * C2
        +   B5 * B4 * A3 * C2
        +   A5 * B4 * A3 * C2
        +   B5 * A4 * A3 * C2
        +   A5 * A4 * A3 * C2
        +   B5 * B4 * A3 * B3
        +   A5 * B4 * A3 * B3
        +   B5 * A4 * A3 * B3
        +   A5 * A4 * A3 * B3
        +   B5 * A4 * B4
        +   A5 * A4 * B4
        +   A5 * B5;
```

Appendix B. PLD ToolKit Code for an Accumulator with Rail Condition (continued)

```
        {A6:  2 product terms, pin 19: 13 P.T. Available}

/A6 =  <XSUM>  C5
      <SUM>  /A6 * /B6
        +      A6 *  B6;

        {A7:  6 product terms, pin 24: 13 P.T. Available}

/A7 =  <XSUM> /A7
      <SUM>   B7 * /B6 * /C5
        +     /B7 *  B6 *  C5
        +     /B7 *  A6 *  C5
        +     /B7 *  A6 *  B6
        +      B7 * /A6 * /C5
        +      B7 * /A6 *  B6;

        {COUT:  7 product terms, pin 18: 17 P.T. Available}

/COUT = <SUM>  /B7 * /B6 * /C5
          +     /A7 * /B6 * /C5
          +     /B7 * /A6 * /C5
          +     /A7 * /A6 * /C5
          +     /B7 * /A6 * /B6
          +     /A7 * /A6 * /B6
          +     /A7 * /B7;

        {R0: rail bit 0; Arbitrarily equation chosen to detect when upper 5 bits are all 1 - this decision is a matter of
preference        output active low}

/R0   = <SUM>  A7 * A6 * A5 * A4 * A3;

        { R1: rail bit 1; Again, arbitrarily chosen to reflect value of carry out, therefore this is a redundant output - active low
        output}

/R1   = <SUM> COUT;

{End of file}
```

CYPRESS SEMICONDUCTOR

FDDI Physical Connection Management Using the CY7C330 Synchronous State Machine

Purpose

The purpose of this application note is to show how the Cypress CY7C330 programmable logic device (PLD) can be used to implement the Physical Connection Management (PCM) state machine specified in the Station Management (SMT) of the Fiber Distributed Data Interface (FDDI) standard. Although there will be a brief overview of the FDDI standard, the purpose of this application note is to show the features of the CY7C330, the design methodology used in this design, as well as an example of how a complex function can be synthesized into this device. This application note is not meant to be an in-depth tutorial of the FDDI standard and its various layers.

FDDI Overview

FDDI is a 100 Mbits/second dual token ring network that can connect as many as 500 nodes with a maximum link-to-link distance of 2 km and a total network circumference of about 100 km. The network employs two rings, a primary and a secondary. The primary ring is for data transmission and the secondary ring is mainly for fault tolerance, but can be used for data transmission as well. It is a token ring network, whereby access is gained to the network by rotating a token. The node with the token can transmit data. This insures a deterministic, collision-free network, independent of the number of stations contained in the network.

Because of the dual ring topology, FDDI defines a fault-recovery mechanism. If a fault is detected, such as a broken fiber-optic cable, the network can be restored by routing around the break with the second ring. This function is largely controlled by the state machine that will be shown later, performed with the CY7C330.

The FDDI standard was developed using the Open Systems Interconnection (OSI) model, implementing the physical and data-link layers of the OSI model. The four FDDI layers are Physical Media Dependent (PMD), Physical (PHY), Media Access Control (MAC), and Station Management (SMT). The PMD layer is the lowest and it specifies the actual connectors, transceivers, and bypass switches. The PHY layer specifies the type of encoding used on the data, 4B/5B, and specifies a set of line states. These line states perform a handshake mechanism between PHYs of adjacent nodes. The MAC layer performs higher-level peer-to-peer communications. It also provides for system timer support, packet framing, and responses to various types of errors in the network. The SMT layer controls the activities of the MAC, PHY, and PMD. It includes functions such as connection management (CMT), fault detection, and ring reconfiguration.

It should be noted that the FDDI standard is controlled by the ANSI X3T9.5 standards committee. At the time of this writing, the committee has accepted the specification of the MAC and PHY layers, and the PMD and SMT specifications are expected to be complete in the summer of 1989. The state machine example specified later was developed with the December 2, 1988 update of the SMT specification. There is a possibility that the final specification might be slightly different, but the design methodology would be the same.

The CMT is the portion of Station Management that controls the insertion, removal and logical connection of the PHY entities. Within the CMT, is an area known as the Physical Connection Management (PCM). A chart showing a hierachical view of the location of the PCM is shown in *Figure* 1. The PCM provides the necessary signals to perform the following functions:

1. initializing a connection
2. reject a marginal connection
3. support maintenance.

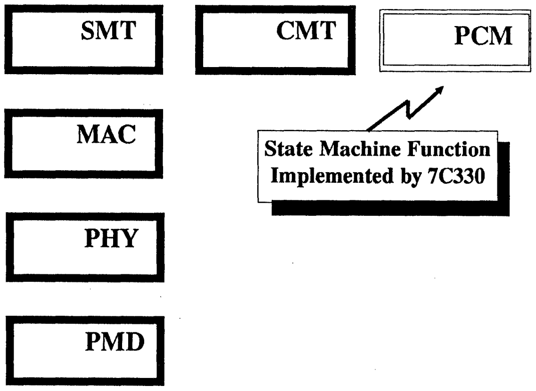


Figure 1. FDDI Hierarchy

The synthesized state machine to perform these activities is show in *Figure* 2 . This state machine is based on version 9.1 of the PCM state machine specified in the SMT specification.

In order to meet the I/O constraints of the CY7C330, of which there are 25 total, there was a small amount of logic that was performed outside the CY7C330. For instance, there are two timers used by the PCM. These timers are not included in the CY7C330, but two signals (timer1 and timer2) are decoded signals that signal that the timer has reached particular values. The signals timer1 and timer2 are inputs to the CY7C330. The chart in *Figure* 3 shows all the macrofunctions, how they are decoded, and their function.

Introduction to the CY7C330

The CY7C330 is a synchronous 28-pin programmable logic device that is packaged in a 300 mil DIP package, as well as several surface mount packages, including leadless ceramic chip carrier (LCC) and plastic leaded chip carrier (PLCC). The device is fabricated on the Cypress 0.8 micron CMOS process, and is available in speeds of 33, 50 and 66 MHz. The device is also available as a military device, in speeds of 33, 40, and 50 MHz. The device is optimized to perform high-speed state machine designs.

The features of the CY7C330 can be generalized into four groups:

1. the dedicated input cell
2. the product term array
3. the I/O macrocell, and
4. the hidden state register macrocell.

The dedicated input cell (see *Figure 4*) contains a D-flip-flop, with a programmable multiplexer (mux) that allows a choice of two input clocks. The two input clocks allowed are CK1 and CK2, which correspond directly to pin 2 and pin 3 of the device, respectively. Note that the input registers (or any other register in the device) are not bypassable. The device is purely synchronous in nature. There are eleven dedicated input macrocells in the device.

The product term array (see *Figure 5 and Figure 6*), as with any programmable logic device, is where the logical connections of the design are synthesized. It contains product terms that control a global reset, a global preset, an exclusive OR gate, the output enables, and the product terms that go to the D input of the flip-flops in the output macrocells. Most of these features will be covered later in the explaination of the macrocell. The device features product term distribution that varies between 9 and 19, depending on which output macrocell is being addressed. The 19 product terms become the limiting factor in the complexity of the design.

The I/O macrocell (see *Figure 7*) contains two D flip-flops. One of the D flip-flops clocks data from the array to either the output pin, or back to the array, and is intended to be a state register. It has a clock, different than the input registers, called CLK, which is derived directly from pin 1. The other D flip-flop is an input register, which can clock data from the I/O pin

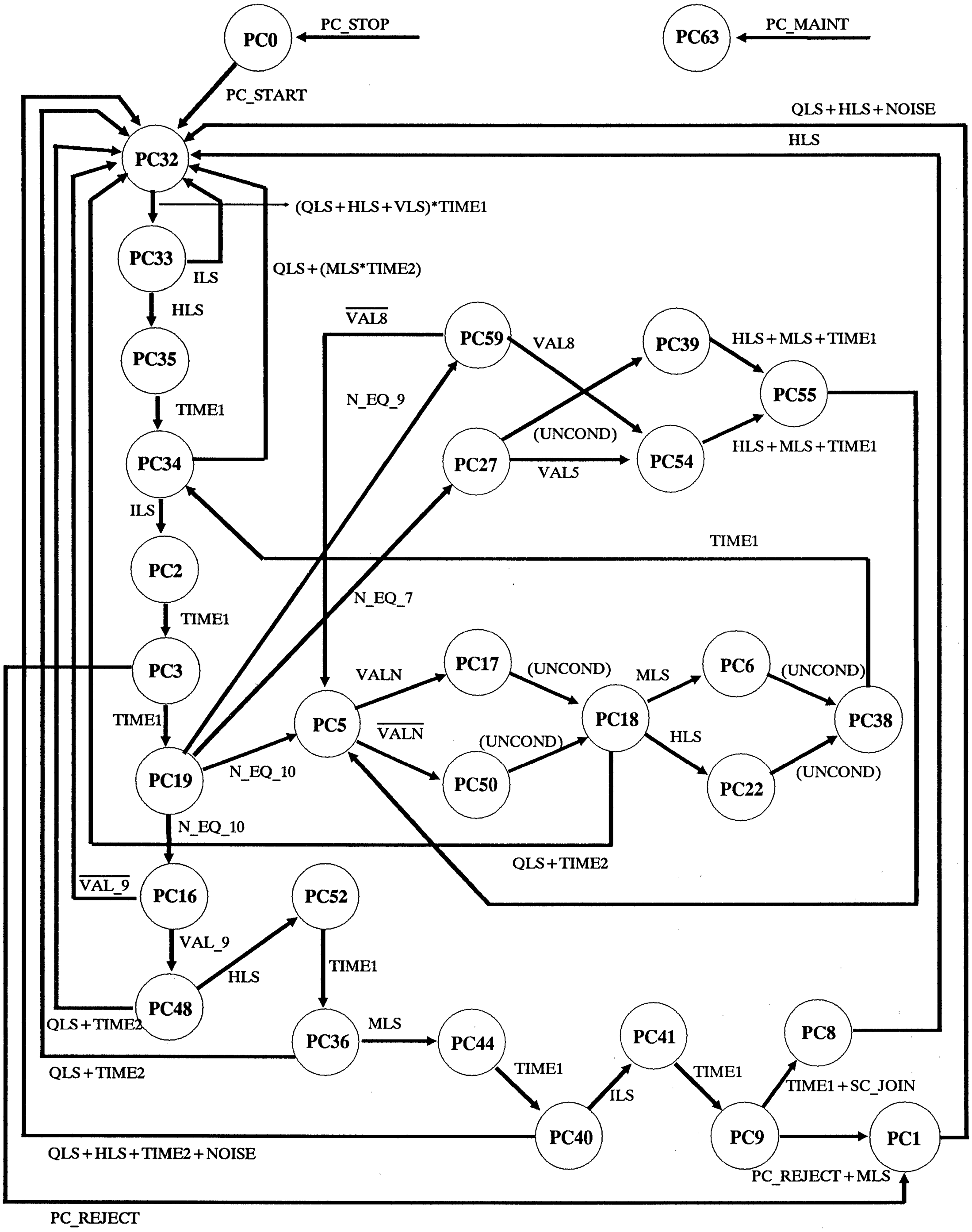


Figure 2. PCM State Machine

MACRO NAME	SYNTHESIZED SIGNAL	FUNCTION
MLS	!MLS	Master Line State
ILS	!ILS	Idle Line State
HLS	!HLS	Halt Line State
QLS	!QLS	Quiet Line State
pc_start	!pc0 & !pc1	State PCM State Machine
pc_reject	!pc0 & pc1	Enter Reject State
sc_join	pc0 & !pc1	Encorporate connection into token path
pcstop	!pc_stop	PCM state machine to enter OFF state
pcmaint	!pc_maint	Enter maintenance state
time1	!timer1	See timer explanations below.
time2	!timer2	See timer explanations below.
n_neq_10	!n0 & !n1	Counter indicating 10 bits of data have not been received or transmitted
n_eq_7	!n0 & n1	Counter indicating 7 bits have been transmitted or received
n_eq_9	n0 & !n1	Counter indicating 9 bits have been transmitted or received
n_eq_10	n0 & n1	Counter indicating 10 bits have been transmitted or received
noise	!noise_count	Noise counter threshold
valn	Val_n	Transmitted value n
val8	!Val_8	Transmitted or Received value = 8
val9	!Val_9	Transmitted or Received value = 9

TIMER VALUES

Timer 1		
0 ms	TB_Min	Minimum break time for link.
0.2 ms	A_Max	Maximum time required to achieve signal aquisition.
480 ns	LS_Min	Length of time reception of ILS
15 us	LS_Max	Max time required for line state recognition
25 ms	I_Max	Max optical bypass insertion/deinsertion time
200 ms	T_next(9)	Default time for MAC loopback
Timer 2:		
100 ms	T_Out	Signalling Timeout

Figure 3. Macro Definitions

into the array. It may be clocked from CK1 or CK2 as in the dedicated input cell. As mentioned previously, there is an XOR gate, fed from the product term array, that feeds the D input of the state register. This gives the designer quite a bit of flexibility. The XOR gate can be used as a simple inverter by setting the XOR product term to a one. The XOR can be used to change the type of the flip-flop from a D to a T, or JK. For example, wrapping the Q output back to the XOR input changes the flip-flop from D-type to T-type. This feature will be used later in the example design. The output macrocell also allows for a choice of the output enable control for the pin. The output enable can be from a product term, or directly from pin 14. There are twelve I/O macrocells in the CY7C330.

The hidden-state macrocell (see *Figure* 8) contains a state register with no output pin associated with it. There are four hidden-state macrocells in the CY7C330. The hidden-state macrocells can be used to synthesize a small 4-bit internal state machine, or perform any function required only internally to the device itself.

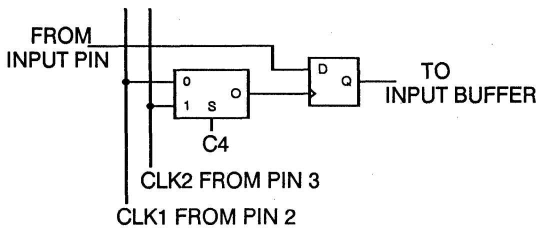


Figure 4. Input Macrocell

Methodology of Design

The PCM design was first attempted using ABEL version 3.0 as a development platform, with the ABEL state machine syntax. The original ABEL source code is shown in *Appendix A*. Note that the state machine requires 31 states. This meant that the state machine could be performed with 5 bits, for 32 total states, leaving one illegal state. When the design was run at reduction level 4, which is the maximum reduction in ABEL, the software responded with the output that the design required in excess of 30 product terms per output. This is far more than the 19 that are possible on any one output. At first glance, one might assume that the design was far too complex for the 7C330. At this point, a process of product term squeezing was initiated. The process is described below.

First of all, a comment on how ABEL performs reduction. ABEL will reduce everything to sum of products, and not make use of the XOR gate in the macrocell. To make use of the XOR gate, you must specify it in boolean equation form, and run the reduction at level 0. Secondly, in ABEL 3.0, specifying T flip-flops will again cause ABEL to reduce to sum of products, and not create the T flip-flop using the XOR gate. ABEL 3.1 accepts T flip-flops and corrects this situation.

The timing required for this design is 12.5 MHz, allowing the slowest version CY7C330, at 33 MHz, to be used. The design requires one clock, although two pins are dedicated for clocks in the CY7C330. In this design, pins 1 and 2 will be tied together externally, making the input registers and state register clock together. The labels for the two clocks in the source code are CKS and CK1.

Product Term Squeezing

The first method of getting the design to use less product terms was to increase the number of bits in the state machine from 5 bits to 6 bits. Although the state machine only requires 31 states, much more choice is allowed for when you have 64 possibilities for placing the states.

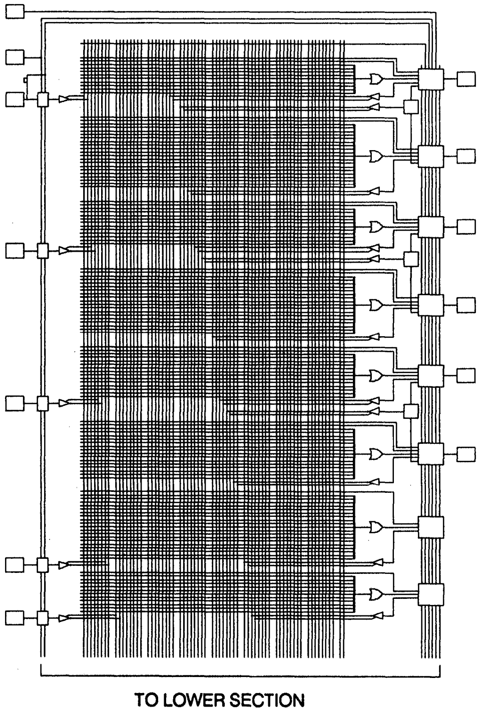


Figure 5. The CY7C330 Block Diagram (Lower Half)

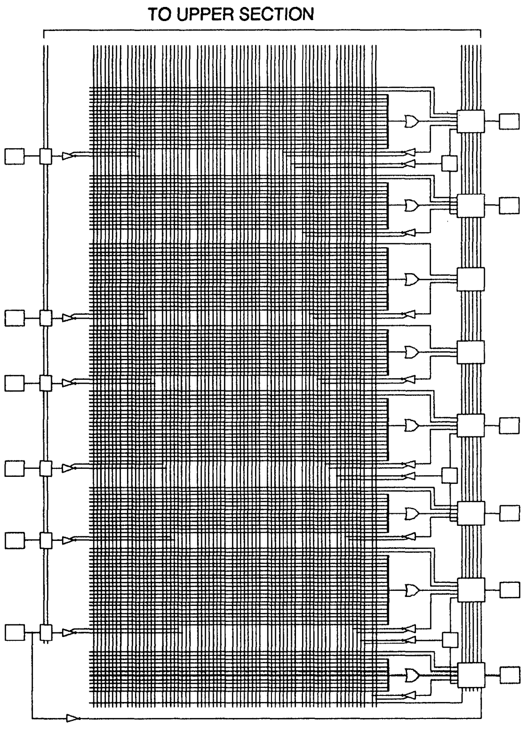


Figure 6. The CY7C330 Block Diagram (Upper Half)

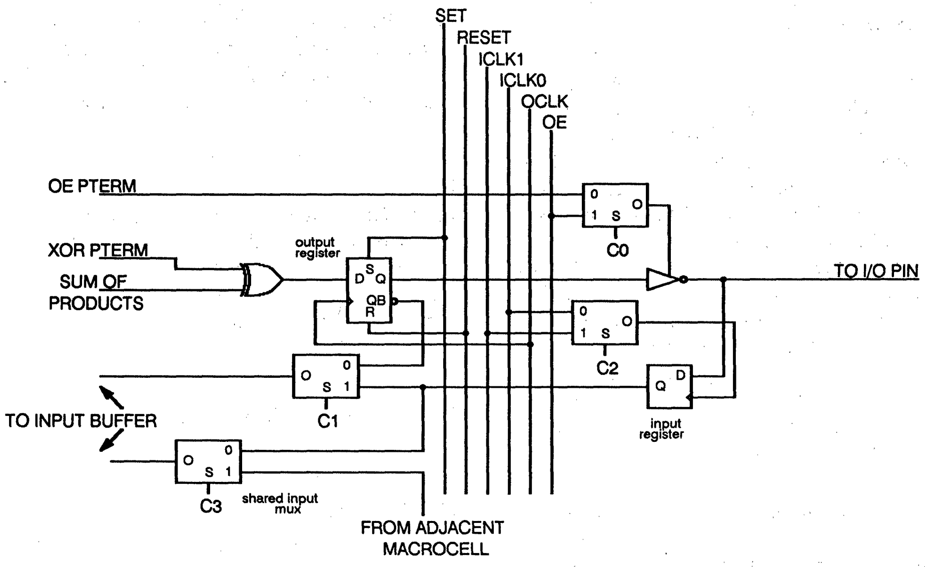


Figure 7. CY7C330 Macrocell

The next procedure involved changing from D flip-flops, to T flip-flops. T flip-flops are more efficient than D flip-flops because when the T input is high, the flip-flop toggles. Otherwise, the flip-flop retains its previous state.

Because a T flip-flop only needs one product term for a transition to occur, the state machine can be optimized by choosing state transitions that use a minimum number of bits. For example, a transition between states 6 and 9 requires more bits to change than a transition between states 6 and 7 as shown in *Figure 8.*

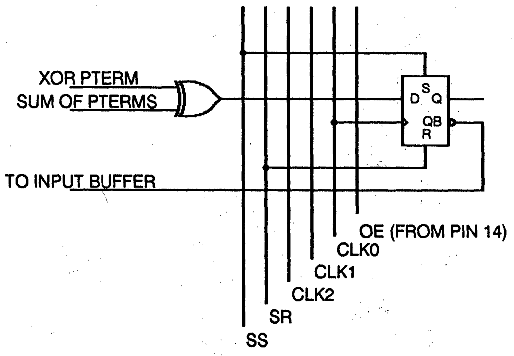


Figure 8. The CY7C330 Buried Register

In case 1, four product terms are required. In case 2, only one product term is required. Since we increased the number of total states available from 32 to 64 by adding one more bit to the state machine, we provide much more flexibility in choosing states. Carefully choosing the states in a state machine is the easiest way to reduce the number of product terms required.

Another way to make the design implementation more efficient is to use the synchronous global reset and preset in the CY7C330. Initially the state machine will be in state 0 because the CY7C330 has a power-on reset. It is good design practice to make provisions for illegal states. Although an illegal state should never occur, the state machine should be able to recover. Many times the recovery mechanism is built into the state machine itself, causing more product terms to be required. In this example, if an illegal state is detected, the state machine will re-initialize itself, and go to state 0. Instead of building this requirement into the design, a hidden register was used to detect the occurrence of illegal states. That signal is then used to control the synchronous reset of the 7C330, which will return the state machine to state 0. Because of the synchronous nature of the device, the state machine will go to state 0 two clocks after the illegal state is encountered. One clock is required to detect the illegal state, and one clock is required to reset the device. This requirement is acceptable for this application.

	Decimal	Binary
Case 1.		
	6	000110
	9	001001
		(4 bits toggle)
Case 2.		
	6	000110
	7	000111
		(1 bit toggles)

Figure 9. State Change Comparison

In this particular design, it was noticed that in every case the condition pcmaint was encountered, the state machine was unconditionally required to go to a particular state. In order to reduce the state machine even further, the state that was chosen on this condition was 63 (111111 binary). The synchronous preset was then used to detect of this signal. When pcmaint is asserted, this forces the state machine to state 63, thus avoiding the use of any product terms in the main body of the design.

In this design, there were several synchronous resets required. There is an external pin (RST), the illegal state detect, and the signal pc_stop. Because there is only one product term allowed for the synchonous reset of the device, the other two resets must be developed by ANDing the reset signal with every product term associated with the outputs that are to be reset. This performs the same function, but does not utilize any additional resources in the CY7C330.

Keep in mind that the CY7C330 has varied product term distribution. The state registers associated with pins 16 and 27 have 19 product terms. Put the state outputs that require the most product terms to these pins. In this example, Q0 required 18 product terms, and Q5 required 17. These outputs were assigned pins 27 and 16. The remaining outputs were placed in the same manner.

Converting the state machine to boolean equations is a straight-forward procedure. By examining the state transistions, the boolean equations can be extracted. The reduced design is shown in *Figure* 10.

The development platform used for which this is the source code is the Cypress PLD ToolKit. The Cypress PLD ToolKit is a low-cost software development system for all Cypress PLD's. Although the reduced equations could have been obtained using ABEL, in many ways the ToolKit is easier to use, and more tailored to the Cypress devices. The ToolKit source file is listed in *Appendix B.* The ToolKit also features a mouse-driven interactive simulator/waveform editor. This makes design verification very easy.

Conclusion

```
State !S48:     if (HLS) then !S52
                else if (QLS # time2) then !S32
                else !S48;

                48 = 110000 (binary)
                52 = 110100
```

Q2 is the only bit that transitions

Therefore, a product term of:

```
        Q5 & Q4 & !Q3 & !Q2 & !Q1 & !Q0 & HLS
        \________________________________/
                     state 48
```

would be added to the equation for Q2.

To continue the example:

```
                48 = 110000
                32 = 100000
```

Q4 is the only bit that transitions

Therefore, the product terms of:

```
          Q5 & Q4 & !Q3 & !Q2 & !Q1 & !Q0 & QLS
        # Q5 & Q4 & !Q3 & !Q2 & !Q1 & !Q0 & time2
          \________________________________/
                       state 48
```

would be added to the equation for Q4.

Figure 10. Boolean Equation Extraction Example

The purpose of this applications note was to introduce the CY7C330, and show a useful application example for the device. Although this example, the PCM state machine for FDDI, is a very complex function, the design was made to fit in the CY7C330. The CY7C330 offers the designer a high degree of flexibility. Using the available software development tools, ABEL and Cypress PLD ToolKit, the designer can implement even more complex functions by following the methodology outlined in this example. There is no other device presently available that can implement complex state machines at the speeds the 7C330 can offer.

Appendix A. Orignal Abel Source Code

```
module pcm flag '-r3'
title 'Physical Connection Management (PCM) state Machine version 9.1
Steve Traum  Cypress Semiconductor  March 27, 1989'

                    U1      device 'P330';
"Inputs
                    CKS,Ck1,rst_            pin 1,2,3;
                    pc0,pc1                 pin 4,5;
                    timer1                  pin 6;
                    timer2                  pin 7;
                    mls,ils,hls,qls         pin 9,10,11,12;
                    Val_n                   pin 13;
                    n0,n1                   pin 14,15;
                    Val_8                   pin 16;
                    Val_9                   pin 17;
                    noise_count             pin 18;
                    pc_stop                 pin 19;
                    pc_maint                     pin 20;
                    n1          istype      'feed_pin';
                    Val_8       istype      'feed_pin';
                    Val_9       istype      'feed_pin';
                    noise_count istype      'feed_pin';

"Outputs

                    Reset                   node 29;
                    Q5,Q4,Q3,Q2,Q1,Q0            pin 28,27,26,25,24,2 3;
                    Q5,Q4,Q3,Q2,Q1,Q0   istype   'pos,reg';
                    Qstate  = [Q5,Q4,Q3,Q2,Q1,Q0];

"declarations
                    High,Low                = 1,0;
                    H,L,C,X,Z         = 1,0, .C.,.X.,.Z.;

"Qstate
S0     = ^b000000;  S1 = ^b000001;   S2 = ^b000010;   S3 = ^b000011; S4 = ^b000100;
S5     = ^b000101;  S6 = ^b000110;   S7 = ^b000111;   S8 = ^b001000;S9 = ^b001001;
S10    = ^b001010;  S11 = ^b001011;  S12 = ^b001100;  S13 = ^b001101;S14 = ^b001110;
S15 = ^b001111;     S16 = ^b010000;  S17 = ^b010001;  S18 = ^b010010;S19 = ^b010011;
S20 = ^b010100;     S21 = ^b010101;  S22 = ^b010 110; S23 = ^b010111;S24 = ^b011000;
S25 = ^b011001;     S26 = ^b011010;  S27 = ^b011011;  S28 = ^b011100;S29 = ^b011101;
S30 = ^b011110;     S31 = ^b011111;  S32 = ^b100000;  S33 = ^b100001;S34 = ^b100010;
S35 = ^b100011;     S36 = ^b100100;  S37 = ^b100101;  S38 = ^b100110;S39 = ^b100111;
S40 = ^b101000;     S41 = ^b101001;  S42 = ^b101010;  S43 = ^b101011;S44 = ^b101100;
S45 = ^b101101;     S46 = ^b101110;  S47 = ^b101111;  S48 = ^b110000;S49 = ^b110001;
S50 = ^b110010;     S51 = ^b110011;  S52 = ^b110100;  S53 = ^b110101;S54 = ^b110110;
S55 = ^b110111;     S56 = ^b111000;  S57 = ^b111001;  S58 = ^b111010;S59 = ^b111011;
S60 = ^b111100;     S61 = ^b111101;  S62 = ^b111110;  S63 = ^b111111;

MLS MACRO {(!mls)};
ILS MACRO {(!ils)};
HLS MACRO {(!hls)};
QLS MACRO {(!qls)};
pc_start MA CRO {(!pc0 & !pc1)};
```

Appendix A. Original Abel Source Code (continued)

```
pc_reject MACRO {(!pc0 & pc1)};
sc_join MACRO {(pc0 & !pc1)};
pcstop MACRO {(!pc_stop)};
pcmaint MACRO {(!pc_maint)};
time1 MACRO {(!timer1)};
time2 MACRO {(!timer2)};
n_neq_10 MACRO {(!n0 & !n1)};
n_eq_7 MACRO {(!n0 & n1)};
n _eq_9 MACRO {(n0 & !n1)};
n_eq_10 MACRO {(n0 & n1)};
noise MACRO {(!noise_count)};
valn MACRO {(Val_n)};
val8 MACRO {(!Val_8 )};
val9 MACRO {(!Val_9)};

state_diagram Qstate
        state !S0:
                if ( pc_start) then !S32
                else if ( pcmaint ) then !S31
                else !S0;
        state !S1:
                if (HLS) then !S32
                else if ( pcstop ) then !S0
                else if ( pcmaint ) then !S63
                else !S1;
        state !S2:
                if (time1) then !S3
                else !S2;
        state !S3:
                if (time1) then !S19
                else if ( pc_reject ) then !S1
                else !S3;
                state !S63:
                if ( pc_stop ) then !S0
                else !S63;
        state !S6:
                goto !S38;
        state !S8:
                if ( QLS # HLS # noise ) then !S32
                else if ( pc_stop ) then !S0
                else if (pc_maint) then !S63
                else if (pc_start) then !S32
                else !S8;
        state !S9:
                if ( sc_join&time1 ) then !S8
                else if ( pc_reject # MLS ) then !S1
                else !S9;
        state !S16:
                if ( val_9 ) then !S48
                else !S32;
        state !S17:
                goto !S18;
```

Appendix A. Original Abel Source Code (continued)

```
state !S18:
        if ( QLS # time2 ) then !S32
        else if ( MLS ) then !S6
        else if (HL S) then !S22
        else !S18;
state !S19:
        if (n_neq_10 ) then !S51
        else if (n_eq_7) then !S27
        else if (n_eq_9) then !S59
        else if (n_eq_10) then !S16
        else !S19;
state !S22:
        goto !S38;
state !S27:
        if (val5 = = High ) then !S54
        else !S39;
state !S39:      if ( HLS # MLS # time1 ) then !S55
        else !S39;
state !S32:      if (( QLS # HLS # MLS) & time1 ) then !S33
        else if ( pc_stop ) then !S0
        else if ( pc_maint) then !S63
        else !S32;
state !S33:
        if ( HLS ) then !S35
        else if (ILS) then !S32
        else !S33;
state !S34:
        if ( ILS ) then !S2
        else if ( QLS # (MLS & time2)) then !S32
        else !S34;
 state !S35:
        if ( time1 ) then !S34
        else !S35
state !S36:
        if ( MLS ) then !S44
        else if ( QLS # time2 ) then !S32
        else if (pc_stop) then !S0
        else if (pc_maint) then !S63
        else !S36;
state !S38:
        if (time1) then !S34
        else !S38;
state !S40:
        if ( ILS ) then !S41
        else if ( QLS # HLS # time2 # noise ) then !S32
        else !S40;
state !S41:
        if ( time1 ) the n !S9
        else !S41;
state !S44:
        if ( time1 ) then !S40
        else !S44;
```

Appendix A. Original Abel Source Code (continued)

```
        state !S48:
                if (HLS) then !S52
                else if (QLS # ti me2) then !S32
                else !S48;
        state !S50:
                goto !S18;
        state !S51:
                if ( valn = = High ) then !S17
                else !S50;
        state !S52:
                if ( time1 ) then !S36
                else !S52;
        state !S55:
                goto !S51;
        state !S59:
:               if (val8) then !S54
                else !S51;
        state !S54:
                if ( HLS # MLS # time1 ) then !S55;
                else !S54;
        state !4        goto !S0;
        state !5:       goto !S0;
        state !7:       goto !S0;
        state !10:      goto !S0;
        state !11:      goto !S0;
        state !12:      goto !S0;
        state !13:      goto !S0;
        state !14:      goto !S0;
        state !15:      goto !S0;
        state !20:      goto !S0;
        state !21:      goto !S0;
        state !23:      goto !S0;
        state !24:      goto !S0;
        state !25:      goto !S0;
        state !26:      goto !S0;
        state !28:      goto !S0;
        state !29:      goto !S0;
        state !30:      goto !S0;
        state !31:      goto !S0;
        state !37:      goto !S0;
        state !42:      goto !S0;
        state !43:      goto !S0;
        state !45:      goto !S0;
        state !46:      goto !S0;
        state !47:      goto !S0;
        state !49:      goto !S0;
        state !53:      goto !S0;
        state !56:      goto !S0;
        state !57:      goto !S0;
        state !58:      goto !S0;
        state !60:      goto !S0;
        state !61:      goto !S0;
        state !62:      goto !S0;
equations
        Reset = !rst_;
end pcm                                         "end of file
```

Appendix B. Cypress PLD ToolKit Source File

```
CY7C330;
 {This file is the Cypress ToolKit Source Code for FDDI Design }

CONFIGURE;

CKS,Ck1,RST_,
pc0, pc1, timer1, timer2, MLS (node=9), ILS, HLS, QLS,
Val_n, n0, n1(iop,ireg), !Q0, Val_8(iop,ireg), !Q1, Val_9(iop,ireg), !Q2,
!Q3 (node=23), noise_count(iop,ireg), !Q4, pc_stop(iop,ireg), !Q5,
pc_maint(iop,ireg), RST, SET, ILSTATE (node=34),

{*********************************************************************}

EQUATIONS;

RST = RST_;

SET = !pc_maint;

ILSTATE =     # Q2 & !Q1 & !Q4 & !Q5 & pc_stop
              # Q2 & Q0 & !Q4 & !Q5 & pc_stop
              # Q1 & Q3 & !Q4 & !Q5 & pc_stop
              # !Q1 & Q2 & Q4 & !Q5 & pc_stop
              # Q0 & Q2 & Q4 & !Q5 & pc_stop
              # Q3 & !Q1 & Q4 & !Q5 & pc_stop
              # Q3 & !Q0 & Q4 & !Q5 & pc_stop
              # Q3 & Q1 & !Q4 & Q5 & pc_stop
              # Q0 & !Q1 & Q2 & !Q4 & Q5 & pc_stop
              # !Q1 & Q3 & Q4 & Q5 & pc_stop
              # !Q1 & Q0 & Q4 & Q5 & pc_stop
              # Q3 & !Q0 & Q4 & Q5 & pc_stop;


Q0  :=        <oe>
              <xsum> Q0 & !ILSTATE & pc_stop
              # !Q5 & !Q4 & !Q3 & !Q2 & !Q1 & Q0 & !HLS & !ILSTATE & pc_stop
              # !Q5 & !Q4 & !Q3 & !Q2 & Q1 & !Q0 & !timer1 & !ILSTATE & pc_stop
              # !Q5 & !Q4 & Q3 & !Q2 & !Q1 & Q0 & pc0 & !pc1 & !timer1 & !ILSTATE & pc_stop
              # !Q5 & Q4 & !Q3 & !Q2 & !Q1 & Q0 & !ILSTATE & pc_stop
              # !Q5 & Q4 & !Q3 & !Q2 & Q1 & Q0 & n0 & n1 & !ILSTATE & pc_stop
              # Q5 & Q4 & Q3 & !Q2 & Q1 & Q0 & !Val_8 & !ILSTATE & pc_stop
              # Q5 & Q4 & !Q3 & Q2 & Q1 & !Q0 & !HLS & !ILSTATE & pc_stop
              # Q5 & Q4 & !Q3 & Q2 & Q1 & !Q0 & !MLS & !ILSTATE & pc_stop
              # Q5 & Q4 & !Q3 & Q2 & Q1 & !Q0 & !timer1 & !ILSTATE & pc_stop
              # !Q5 & Q4 & Q3 & !Q2 & Q1 & Q0 & Val_n & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & !Q2 & !Q1 & !Q0 & !QLS & !timer1 & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & !Q2 & !Q1 & !Q0 & !HLS & !timer1 & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & !Q2 & !Q1 & !Q0 & !MLS & !timer1 & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & !Q2 & !Q1 & Q0 & !ILS & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & !Q2 & Q1 & Q0 & !timer1 & !ILSTATE & pc_stop
              # Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !ILS & !ILSTATE & pc_stop
              # Q5 & Q4 & !Q3 & !Q2 & Q1 & Q0 & !Val_n & !ILSTATE & pc_stop
              # Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & !pc0 & !pc1 & !ILSTATE & pc_stop;
```

Appendix B. Cypress PLD ToolKit Source File (continued)

```
Q1  :=        <oe>
              <xsum> Q1 & !ILSTATE & pc_stop
              # !Q5 & !Q4 & !Q3 & !Q2 & Q1 & Q0 & !pc0 & pc1 & !ILSTATE & pc_stop
              # Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & !pc0 & !pc1 & !ILSTATE & pc_stop
              # !Q5 & Q4 & !Q3 & !Q2 & !Q1 & Q0 & !ILSTATE & pc_stop
              # !Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !QLS & !ILSTATE & pc_stop
              # !Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !timer2 & !ILSTATE & pc_stop
              # !Q5 & Q4 & !Q3 & !Q2 & Q1 & Q0 & n0 & n1 & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & !Q2 & !Q1 & Q0 & !HLS & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & !Q2 & Q1 & !Q0 & !QLS & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & !Q2 & Q1 & !Q0 & !timer2 & !MLS & !ILSTATE & pc_stop
              # Q5 & Q4 & !Q3 & !Q2 & Q1 & Q0 & Val_n & !ILSTATE & pc_stop;


Q2  :=        <oe>
              <xsum> Q2 & !ILSTATE & pc_stop
              # Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & !pc0 & !pc1 & !ILSTATE & pc_stop
              #!Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !HLS & !ILSTATE & pc_stop
              # !Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !MLS & !ILSTATE & pc_stop
              # Q5 & Q4 & Q3 & !Q2 & Q1 & Q0 & !Val_8 & !ILSTATE & pc_stop
              # !Q5 & Q4 & Q3 & !Q2 & Q1 & Q0 & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & Q2 & !Q1 & !Q0 & !QLS & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & Q2 & !Q1 & !Q0 & !timer2 & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & Q2 & Q1 & !Q0 & !timer1 & !ILSTATE & pc_stop
              # Q5 & !Q4 & Q3 & Q2 & !Q1 & !Q0 & !timer1 & !ILSTATE & pc_stop
              # Q5 & Q4 & !Q3 & !Q2 & !Q1 & !Q0 & !HLS & !ILSTATE & pc_stop
              # Q5 & Q4 & !Q3 & Q2 & Q1 & Q0 & !ILSTATE & pc_stop;


Q3  :=        <oe>
              <xsum> Q3 & !ILSTATE & pc_stop
              # Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & !pc0 & !pc1 & !ILSTATE & pc_stop
              # !Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !QLS & !ILSTATE & pc_stop
              # !Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !HLS & !ILSTATE & pc_stop
              # !Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !noise_count & !ILSTATE & pc_stop
              # !Q5 & !Q4 & Q3 & !Q2 & !Q1 & Q0 & !pc0 & pc1 & !ILSTATE & pc_stop
              # !Q5 & !Q4 & Q3 & !Q2 & !Q1 & Q0 & !MLS & !ILSTATE & pc_stop
              # !Q5 & Q4 & !Q3 & !Q2 & Q1 & Q0 & !n0 & n1 & !ILSTATE & pc_stop
              # !Q5 & Q4 & !Q3 & !Q2 & Q1 & Q0 & n0 & !n1 & !ILSTATE & pc_stop
              # Q5 & Q4 & Q3 & !Q2 & Q1 & Q0 & !ILSTATE & pc_stop
              # Q5 & !Q4 & !Q3 & Q2 & !Q1 & !Q0 & !MLS & !ILSTATE & pc_stop
              # Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !QLS & !ILSTATE & pc_stop
              # Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !HLS & !ILSTATE & pc_stop
              # Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !timer2 & !ILSTATE & pc_stop
              # Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !noise_count & !ILSTATE & pc_stop;
```

Appendix B. Cypress PLD ToolKit Source File (continued)

```
Q4  :=          <oe>
                <xsum> Q4 & !ILSTATE & pc_stop
                # !Q5 & !Q4 & !Q3 & !Q2 & Q1 & Q0 & !timer1 & !ILSTATE & pc_stop
                # Q5 & Q4 & Q3 & Q2 & Q1 & Q0 & !pc0 & !pc1 & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & !Q2 & !Q1 & !Q0 & Val_9 & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !QLS & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !timer2 & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !MLS & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & Q2 & Q1 & !Q0 & !ILSTATE & pc_stop
                # !Q5 & Q4 & Q3 & !Q2 & Q1 & Q0 & !Val_n & !ILSTATE & pc_stop
                # Q5 & !Q4 & !Q3 & Q2 & Q1 & Q0 & !HLS & !ILSTATE & pc_stop
                # Q5 & !Q4 & !Q3 & Q2 & Q1 & Q0 & !MLS & !ILSTATE & pc_stop
                # Q5 & !Q4 & !Q3 & Q2 & Q1 & Q0 & !timer1 & !ILSTATE & pc_stop
                # Q5 & Q4 & !Q3 & !Q2 & !Q1 & !Q0 & !QLS & !ILSTATE & pc_stop
                # Q5 & Q4 & !Q3 & !Q2 & !Q1 & !Q0 & !timer2 & !ILSTATE & pc_stop
                # Q5 & Q4 & !Q3 & Q2 & !Q1 & !Q0 & !timer1 & !ILSTATE & pc_stop;

Q5  :=          <oe>
                <xsum> Q5 & !ILSTATE & pc_stop
                # !Q5 & !Q4 & !Q3 & !Q2 & !Q1 & !Q0 & !pc0 & !pc1 & !ILSTATE & pc_stop
                # !Q5 & !Q4 & !Q3 & !Q2 & !Q1 & Q0 & !HLS & !ILSTATE & pc_stop
                # !Q5 & !Q4 & !Q3 & Q2 & Q1 & !Q0 & !ILSTATE & pc_stop
                # !Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !QLS & !ILSTATE & pc_stop
                # !Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !HLS & !ILSTATE & pc_stop
                # !Q5 & !Q4 & Q3 & !Q2 & !Q1 & !Q0 & !noise_count & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & !Q2 & !Q1 & !Q0 & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !QLS & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !timer2 & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & !Q2 & Q1 & Q0 & !n0 & !n1 & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & !Q2 & Q1 & Q0 & n0 & !n1 & !ILSTATE & pc_stop
                # !Q5 & Q4 & !Q3 & Q2 & Q1 & !Q0 & !ILSTATE & pc_stop
                # !Q5 & Q4 & Q3 & !Q2 & Q1 & Q0 & !ILSTATE & pc_stop
                # Q5 & !Q4 & !Q3 & !Q2 & Q1 & !Q0 & !ILS & !ILSTATE & pc_stop
                # Q5 & !Q4 & Q3 & !Q2 & !Q1 & Q0 & !timer1 & !ILSTATE & pc_stop
                # Q5 & Q4 & !Q3 & !Q2 & Q1 & !Q0 & !ILSTATE & pc_stop
                # Q5 & Q4 & !Q3 & !Q2 & Q1 & Q0 & Val_n & !ILSTATE & pc_stop;

{end of file}
```

CYPRESS SEMICONDUCTOR

CY7C331 Application Example: Asynchronous, Self-Timed VME Bus Requester

Introduction

This application note shows the capability of the Cypress CY7C331 CMOS Erasable Programmable Logic Device (EPLD) to support asynchronous, self-timed designs. The CY7C331 is ideal for implementation of asynchronous, self-timed, and general-purpose logic integration applications. The ability to implement self-timed applications is unique to the CY7C331. The application example shown consists of the design and implementation of a self-timed VME Bus Requester.

The CY7C331 is a member of the Cypress slimline 28-pin family of high-performance CMOS EPLDs. Family members are characterized by high speed, increased I/O, and high integration. The CY7C331 has a highly flexible architecture intended to support asynchronous and general-purpose logic integration applications. The device has a 192 product term logic array and twelve I/O logic macrocells. Each macrocell has two D-type flip-flops with asynchronous set, reset, and bypass capability. The clock, set and reset inputs of a flip-flop are individually programmable. Logic polarity and output enable control are also individually programmable in each macrocell. Combinatorial and registered inputs and outputs and buried states are easily supported by the CY7C331.

The CY7C331 has the unique capability to be able to self-time asynchronous, sequential applications. A self-timed design performs a sequential task without the presence of a clock to synchronize each step in the sequence. The benefit of this design approach is usually higher performance. The main application for self-timing is in high-performance I/O interfaces. No other PLD has this capability. The CY7C331 is able to support self-timed designs because clock inputs are programmable, internal timing relationships are well-controlled, and metastable resolution is ultra-fast.

The VME Bus Requester application example shows the CY7C331 in an asynchronous, self-timed design. The VME Bus is a common, high-performance asynchronous bus. The VME bus request function is asynchronously initiated and sequential. The application example also shows usage of many of the features of the CY7C331.

CY7C331 Brief Description

The CY7C331 is a member of the Cypress slimline 28-pin family of CMOS, UV-erasable programmable logic devices. The device is available in a 28-pin slimline (.3-inch wide) plastic or windowed DIP, and 28-pin PLCC and LCC packages. The windowed DIP version of the device is erasable and reprogrammable, and the plastic Dip, PLCC, and LCC versions are one-time programmable. The CY7C331 is available with TPD and TCO specified as a maximum of 25 ns, with register set-up times of 12 or 2 ns, depending on whether the register is connected to an input pin or to the array. Other commercial and military speed grades are available.

The CY7C331 is based on a programmable sum-of-products (AND-OR) logic array architecture. The logic array consists of 192 programmable product terms, each having as input the true and complement versions of thirty-one logic inputs. The product terms connect to one of twelve I/O logic macrocells, each connecting to a device pin. The product terms are allocated with a variable distribution to the macrocells. There are thirteen combinatorial inputs to the array from dedicated input

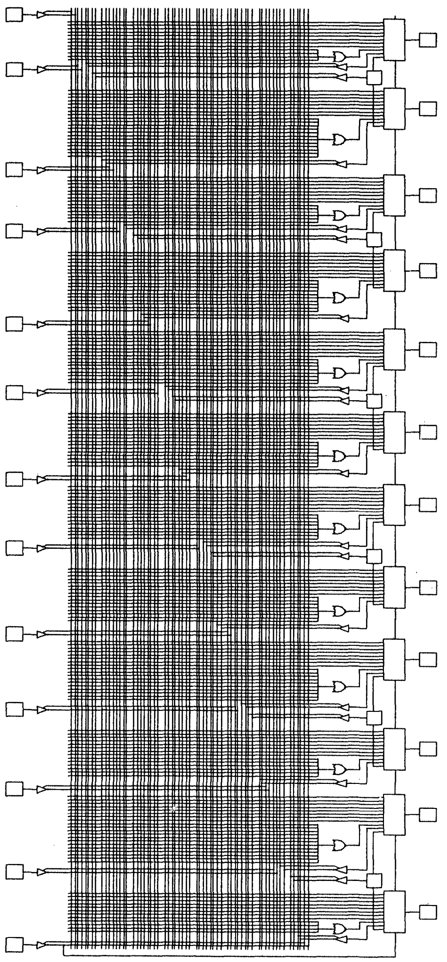


Figure 1. The CY7C331 Block Diagram

pins, one of which (pin 14) may also be used as an output enable control. The macrocells and six shared input muxes each provide an input to the array. A shared input mux selects the input from one of two adjacent macrocells. (Refer to *Figure 1.*)

An I/O macrocell sums array product terms, selectively inverts the sum and provides the result to the D-input of a D-type flip-flop. The output (Q) of the flip-flop is connected through an inverting tri-state buffer to a device pin and can be fed back to the array. An I/O macrocell also provides a second D-type flip-flop that latches data from the same device pin. The Q output of this flip-flop connects to the array input select mux and to the shared input mux (see *Figure 2*). Both flip-flops have asynchronous set (S) and reset (R) inputs, and bypass capability. A flip-flop will bypass the D input to Q when S and R are both high. The clock, S, and R inputs of both flip-flops are each driven from separate product terms.

A multi-input OR gate sums the product terms. The number of product terms input to the OR gate depends on the macrocell (see *Figure 1*). A dual-input XOR gate selectively inverts the sum. The second input of the XOR gate is a product term that can be used to control polarity, or to emulate T or JK type flip-flops. The output enable of a macrocell can be controlled by pin 14, or a product term. One of these two options is selected by the OE mux. The macrocell array input is selected by another mux called the feedback mux. Each OE, feedback and shared input mux has an associated programmable configuration bit that controls mux selection.

CY7C331 Self-Timed Capability

The CY7C331 is designed with the capability to implement self-timed designs. The main application for self-timed functions are in high-performance I/O interfaces where clocking restrictions prevent performance requirements from being satisfied. These applications may not have an available clock, the clock may be too slow or synchronization time may have to be minimized.

A self-timed design implements a state machine without the presence of a clock to synchronize each state transition. The implementation of a self-timed design must meet two basic requirements:

1. Time and perform state transitions.
2. Synchronize asynchronous inputs.

As in any state machine, a self-timed design must meet minimum state flip-flop set up times before performing a state transition. Without the benefit of a clock, self-timing clocks must be generated based on the state data change due to a state transition itself. This means that clock initiation and data changes are coincident. A clock must be delayed to allow data to settle and meet minimum set up time requirements. The simplest example of self-timing is shown in *Figure 3*. A logic one is clocked into a D-type flip-flop on the rising edge of the input. The design works if the clock delay time is long

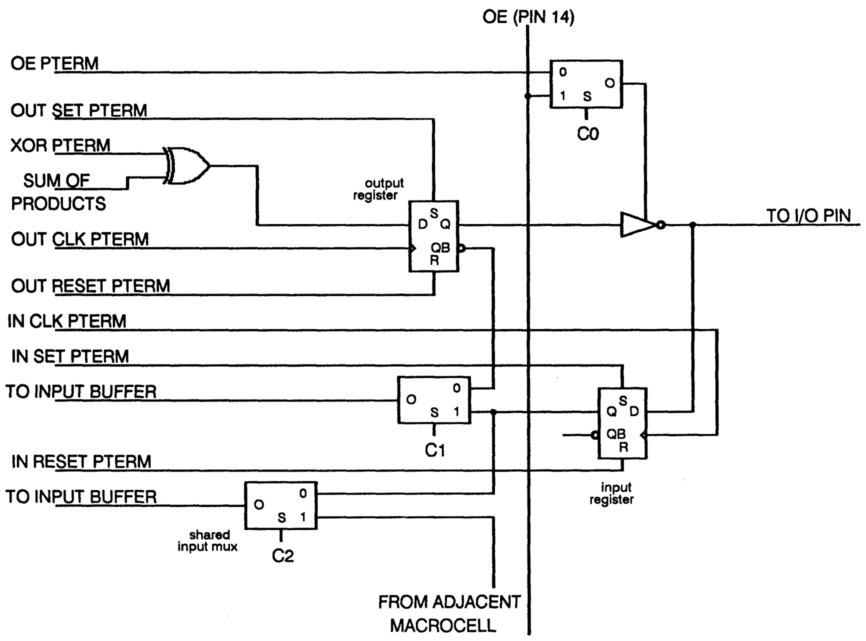


Figure 2. The CY7C331 I/O Macrocell

enough to allow the data input to be set up. The CY7C331 is able to support self-timed designs because the timing relationship between the D input logic and clock input logic of a flip-flop can be programmed to guarantee that minimum set up time requirements are satisfied. The synchronization of asynchronous inputs is performed in the same manner, except that set up time is longer to allow for metastable resolution. The CY7C331 can also perform self-timed synchronization because metastable resolution is ultra-fast.

The approach used in the CY7C331 to self-time state transitions is to delay a clock signal by passing it through the logic array one additional time to allow data to meet set up time requirements. Further, to guarantee that this approach works, the extra level of delay in the clock path must be programmed to delay the clock as long as possible (see *Figure 4*). In general, a self-timed design should set up data as fast as possible and delay the clock long enough to guarantee that data is set up. Delay time in the CY7C331 is sensitive to the logic function programmed. To guarantee that data is set up as fast as possible would restrict logic functions that could be performed. This is avoided by placing restrictions on the clock path. Any logic function can be programmed when the clock delay path element is as slow as possible.

To perform self-timed synchronization, the clock is delayed by two extra passes to allow for the extra delay required for metastable resolution (see *Figure 5*). Both clock delay elements must be programmed to be as slow as possible to allow any logic function to be programmed. These restrictions allow for a Mean Time to Failure (MTF) of greater than 10 years due to a metastable condition in a CY7C331.

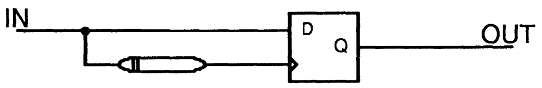


Figure 3. A Simple Self-Timed Element

Clock Delay Programming

In the CY7C331, a product term output transition from low to high is generated faster than from high to low. A transition caused by a single input and a single product term will be faster than those caused by multiple inputs or product terms. The shortest delay time through a

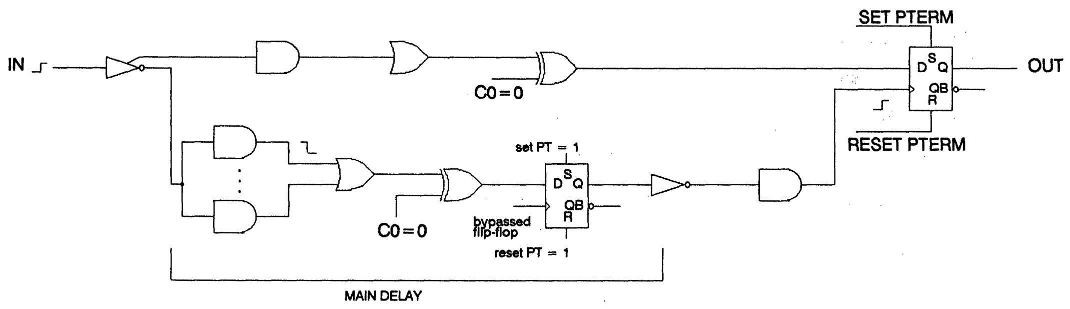


Figure 4. CY7C331 Self-Timed Element

CY7C331 is when a single input triggers a single product term to transition from low to high. The slowest clock path is obtained by placing restrictions on how the extra level of clock delay is programmed. These restrictions are:

1. The clock delay should use a multiple product term, OR gate, XOR gate logic path to a bypassed flip-flop.

2. Clock delay logic should make product term outputs transition from high to low.

3. All product terms to the OR gate should be programmed identically to implement clock logic. The OR gate should have the same or more inputs than associated data path OR gates.

4. The programmable XOR input should be set always low.

The clock delay element of *Figure 4* illustrates each of the four programming restrictions.

Self-Timed VME Bus Requester

The application used to illustrate the use of the CY7C331 in a self-timed design is a VME bus requester. Bus requesters are used in common bus systems that support multiple processors controlling bus transfers. A processor that controls bus transfers is typically referred to as a bus master. The function of a bus requester is to request permission for a master to control data bus transfers. The requester also indicates to the master when control has been granted. The VME Bus is a common, high-performance asynchronous bus that supports multiple bus masters.

A self-timed design approach for a VME bus requester is appropriate because the VME bus is asynchronous and high performance. The bus request function is asynchronously initiated and sequential. A self-timed design will self-synchronize to initiate the request and self-time the rest of the request sequence at CY7C331 device speed. A synchronous approach requires an external clock to synchronize and time the sequence. The VME bus provides a 16 MHz system clock. A

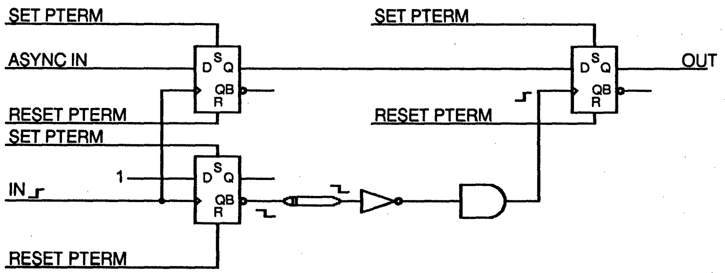


Figure 5. CY7C331 Self-Synchronizing Element

CY7C331 self-timed design provides much higher performance than a synchronous design using the system clock.

The application example also shows usage of many of the features of the CY7C331, and the process used for design and implementation with a CY7C331. The VME bus requester design supports request generation for three on-board masters and overlaps requests with bus transfers. The requester is assumed to be on a board that contains three separate DMA channels. Each channel is a bus master. The requester prioritizes on-board grants to the three masters. A bus master must obtain the bus before data transfers can be performed. This is extra overhead that can lower bus performance. The requester is designed so that bus requests are overlapped with data bus transfers to maintain high performance. The features of the CY7C331 allow these additional functions to be implemented into the requester.

VME Background

The VME bus is defined to support multiple bus masters. Only one bus master can control the bus at a time. The VME bus provides an arbitration subsystem to allocate the data bus. A central bus arbiter determines which master is granted the data bus. Each master contains a bus requester to request control of the bus from the arbiter.

The arbitration subsystem is supported on the bus with six bused lines and four daisy-chained lines. All of these lines are active low; indicated by a '-' suffix on a line name. The bused lines are Bus Busy (BBSY-), Bus Clear (BCLR-), and Bus Request 3-0 (BR3- - BR0-). The daisy-chained lines when entering a board are designated Bus Grant 3-0 In (BG3IN- - BG0IN-) and when leaving are designated Bus Grant 3-0 Out (BG3OUT- - BG0OUT-). (The terms BRx-, BGxIN-, and BGxOUT- are used when references aren't to a specific line or lines. *x* is assumed to be any value from 0 to 3.) Highest priority is allocated to number 3 lines and lowest 0 lines. The BGxOUT- lines that leave a board in slot *n* enter the board in slot *n + 1* as BGxIN- lines. The bus arbiter must always reside in the first slot of a VME bus-based system to initiate BGxOUT- generation.

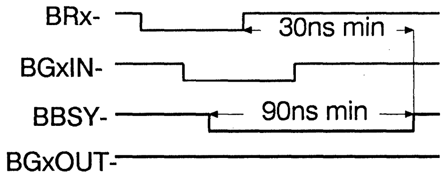


Figure 6. VME Arbitration Timing

A simple VME Bus requester initiates a request when an on-board request (OBR) has been detected. (A simplified bus request state diagram and timing diagram appear in *Figures 6* and *7*.) The requester then drives the appropriate BRx- line active and waits for the associated BGxIN- line to become active. Once the re-

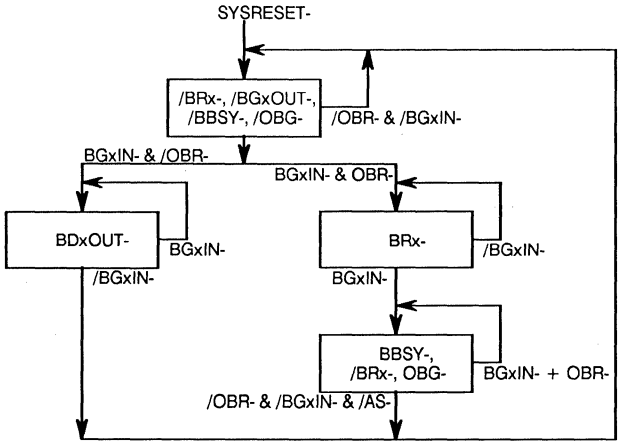


Figure 7. VME Bus Requester State Diagram

quester detects BGxIN- active, BBSY- and an on-board grant (OBG) are driven active and BRx- is released to inactive. OBG indicates to a master that it has the bus and may perform a data transfer once the previous transfer has completed. Transfer completion is indicated when the Address Strobe (AS-) is inactivated. The requester releases the bus by releasing BBSY- and OBG when BGxIN- and OBR have become inactive. If BGxIN- becomes active, but OBR isn't, the requester passes the grant down the daisy chain by making BGxOUT- active.

A VME bus requester must meet two timing requirements. BBSY- must be driven for a minimum of 90 nanoseconds and release of BRx- must occur at least 30 nanoseconds before BBSY- is released. BGxOUT- must never glitch during operation. BBSY- and BRx- lines must use open-collector drivers. All masters drive BBSY- and all masters on a bus grant daisy chain drive the same BRx- line. More than one master on a bus grant daisy chain may request the data bus at the same time by simultaneously driving their associated BRx- line.

Requester Design

The first concern of the design is to understand the functions of the example requester. The requester is defined to support overlapped bus requests and release the data bus every transfer cycle. The data bus is released each transfer cycle because the extra overlapped bus arbitration performance overhead is small and requester design is simplified. The requester supports three on-board DMA request lines (DMARQ2- - DMARQ0-). All of the DMARQx- lines must be able to generate a bus request on the BRx- line. The requester supports three on-board grant lines (DMAGR2- - DMAGR0-), one for each request line. When a bus grant is received on BGxIN-, the requester must determine which DMAGRx- line to activate. The requester prioritizes the DMARQx- lines and grants to the highest priority request. DMARQ0- has the highest priority and DMARQ2- has the lowest. The selected DMAGRx- line must not be activated until the previous data transfer is complete.

The requester must drive BBSY- to take control of the data bus. If any of the DMARQx- lines is requesting the bus when a grant is received, the requester will drive BBSY-. To support overlapped operation, BBUSY- is released as soon as possible to facilitate the next bus arbitration. BBSY- must be driven for at least 90 nanoseconds, and until BGxIN- is released and the previous data transfer is complete. If none of the DMARQx- lines is requesting the bus when a grant is received, the requester must pass the grant onto BGxOUT- for the next requester on the daisy chain. The requester must also recognize a system reset (SYSRESET-).

A logic diagram of a self-timed implementation of the example VME bus requester using the CY7C331 appears in *Figure 8.* BRx- is the OR of the DMARQx- lines. If any DMARQx line becomes active, BRx will become active. BRx is driven by an external inverting open-collector driver.

Self-timed operation is initiated by the incoming BGxIN- line becoming active. The three on-board DMA request lines (DMARQ2- - DMARQ0-) are self-synchronized to the BGxIN- line. The falling edge of BGxIN- is used as a clock to register the DMARQx- lines and toggle a flip-flop from high-to-low to initiate an internal, self-timed clock signal (STCP). The DMARQx- lines must be synchronized because BGxIN- can be activated by any BRx- becoming active or BBSY- being released. For example, if DMARQ0 caused the associated BRx- to initiate bus arbitration, and DMARQ2 attempted to become active at the same time BGxIN- became active, the resulting state of DMARQ2 could be an indeterminate metastable that would need time for resolution.

The internal, self-timed clock signal is delayed by two of the CY7C331 delay elements to allow for the time required to self-synchronize the requests. The requests are prioritized during the clock delay time. The resulting delayed clock (STCP2) then clocks a NOR of the requests into a register to generate BBSY-, and an OR of the requests to generate BGxOUT-. This guarantees that both lines are synchronized and won't glitch.

BBSY- is driven onto the bus with an external inverting open- collector driver. The prioritized requests are clocked into registers to create the DMAGRx- signals on the rising edge of the delayed STCP if the previous data transfer had already completed, or on the rising edge of AS- when the data transfer completes. An internal flip-flop toggles at the same time. The flip-flop output is used to indicate transfer completion (TC).

The registered BBSY- line is fed into an external 90 ns line that is used to guarantee that BBSY- will be active for the minimum required time. BBSY- is inactivated when the 90 ns delay has elapsed, TC is indicated and

BGxIN- is inactive. The requester is initialized for another self-timed operation at the same time. The requester is initialized by clearing the STCP and TC flip-flops.

A SYSRESET resets the requester by clearing the DMAGRx- lines and generating a requester initialize. The design assumes that the 30 ns minimum release time requirment for BRx- before BBSY- is done externally. This gives the DMA masters flexibility to determine operation of their own request lines.

The delay line is used in the design because an absolute delay is required to meet the VME specification. A self-timed delay can yield only relative results. There is no way to determine how many levels of delay would be required to obtain a 90 ns delay. Any one delay is usually much faster than worst-case, but may be that slow. The delay can be emulated on-chip by creating a digital delay, but accuracy would be poor because BBSY- would have to be synchronized to an absolute time base such as the 16 MHz system clock. The external inverting open-collector drivers can be emulated by the CY7C331, but they wouldn't meet the drive requirements of the VME bus specification. Emulation of an open-collector driver requires that the signal output to the external driver instead be used to drive the OE of an on-board inverting tristate driver (with the input tied high).

CY7C331 Implementation

The implementation of the example design was specified for assembly and simulation by the ABEL™ PLD design support software package. The ABEL package, like any other PLD software, allows a design to be specified in terms of boolean equations and automatically generates the appropriate programming pattern for a selected PLD to implement the design. PLD software also typically provides simulation capability to verify correct design operation.

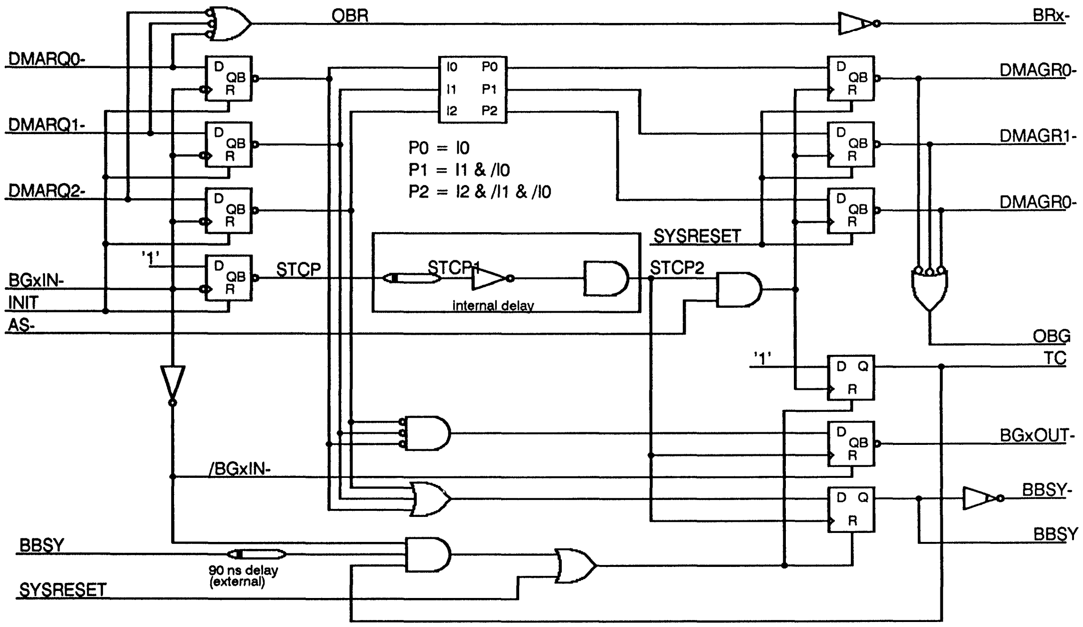


Figure 8. Self-Timed VME Bus Requester Schematic

The DMARQx- lines are defined to use two CY7C331 pins for each line; one combinatorial and one registered. The registered input pins are used to conserve output logic for other functions. The three macrocells associated with the registered inputs also are used to perform the internal self-timed clock generation and delay functions: Most other PLDs require six outputs to implement these functions. In addition, the individually programmable clocks of the CY7C331 allow the input register flip-flops to be clocked on the falling edge of BGxIN-.

The BR and BBSY lines are defined to be active high to allow for external inverting open-collector drivers. BBSY is assumed to be the input to the external delay line and the CY7C331 input BBSY90 is assumed to connect to the delay line output.

The self-timed clock generation and delay logic is defined to meet the requirements of CY7C331 self-synchronization. The ABEL source file for this implementation are available upon request.

CYPRESS SEMICONDUCTOR

Bus-Oriented Maskable Interrupt Controller

Introduction

In virtually all microprocessor designs there is a requirement for some level of interrupt support. In complex applications, a dedicated interrupt controller chip from the specific microprocessor family can provide the required support, but for simple applications, or where special requirements exist, a standard interrupt controller is either inadaquate or represents overkill for the design. In such cases, a custom-designed controller is implemented using some combination of MSI logic and PLDs. This application note is intended to illustrate the design flexibility of the CY7C331 PLD from Cypress Semiconductor in a single-chip interrupt controller design. The design is implemented in two stages. The first is a simple 4-channel controller where the major functional blocks are developed. In the second stage, the simple design is extended to allow cascading of a second controller to provide support for up to 8 interrupt channels.

Design features of the interrupt controller include:

1. Programable Polarity Level Sensitive Inputs
2. Interlocked REQ/ACK Handshake
3. Simple MPU Bus Attachment For Read and Write
4. Masking of Individual Channels
5. Prioritized Interrupt Vector
6. Fully Asynchronous Operation

Description

The interrupt controller is attached to the MPU data bus and is controlled by the system processor through a Read and a Write port on the data bus. The Read port provides interrupt status and a prioritized vector for the processor and the Write port allows the processor to selectively mask indiviual interrupt channels. A separate interrupt request line to the processor is provided to signal a pending interrupt. The bit assignments for the Read and Write ports are defined in *Figure 1.*

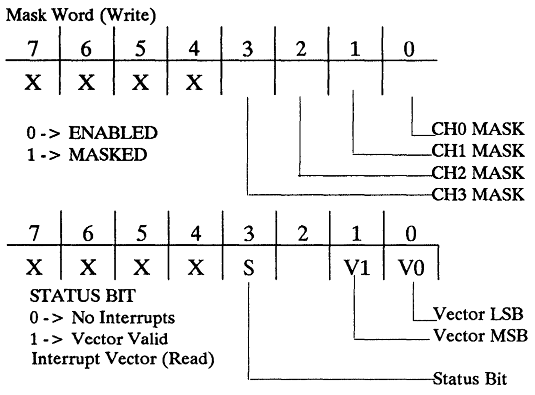


Figure 1. Data Bus Bit Assignments

A functional block diagram of the interrupt controller is provided in *Figure 2.* Major functional blocks include the Mask Register and Gating block, the Priority Encoder and Latch, and the Acknowledge Generator block.

The operation of the interrupt controller is quite simple. On reset, all interrupt channels are masked off, and no interrupts are permitted. The processor then loads the mask register with the desired interrupt channel mask bits cleared. When an interrupt request oc-

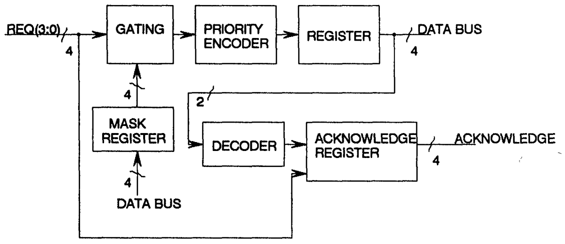


Figure 2. Interrupt Controller Block Diagram

curs, if the channel is not masked, the request is prioritized and the Interrupt Request (IRQ) to the processor is asserted. The processor responds to the IRQ by reading the Interrupt Vector port. When the read is detected by the interrupt controller, the current interrupt priority is latched and the priority vector is placed on the data bus. Latching the current priority while the vector is being read prevents the vector from being altered in the midst of the read cycle. In addition, the vector is decoded within the interrupt controller and the Acknowledge line of the corresponding channel is asserted. The Acknowledge remains asserted until detected by the interrupting element, which

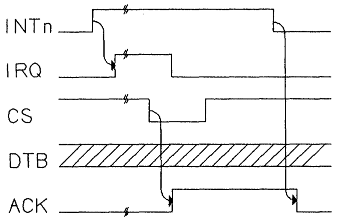


Figure 3. Timing Sequence for Single Interrupt Channel

responds by deasserting its interrupt request. This interlocking handshake insures that a pending interrupt is not lost or responded to more than once. The Acknowledge is also used internally to disable the interrupt request into the priority encoder during the interval between the interrupt acknowledge and the interrupt request being deasserted. A simple example of the timing sequence for a single interrupting channel is provided in *Figure* 3.

CY7C331 Description

The CY7C331 is an asynchronous PLD packaged in a 28 pin 300 mil DIP. The device features 12 I/O macrocells and 13 dedicated inputs. The I/O macrocell is con-

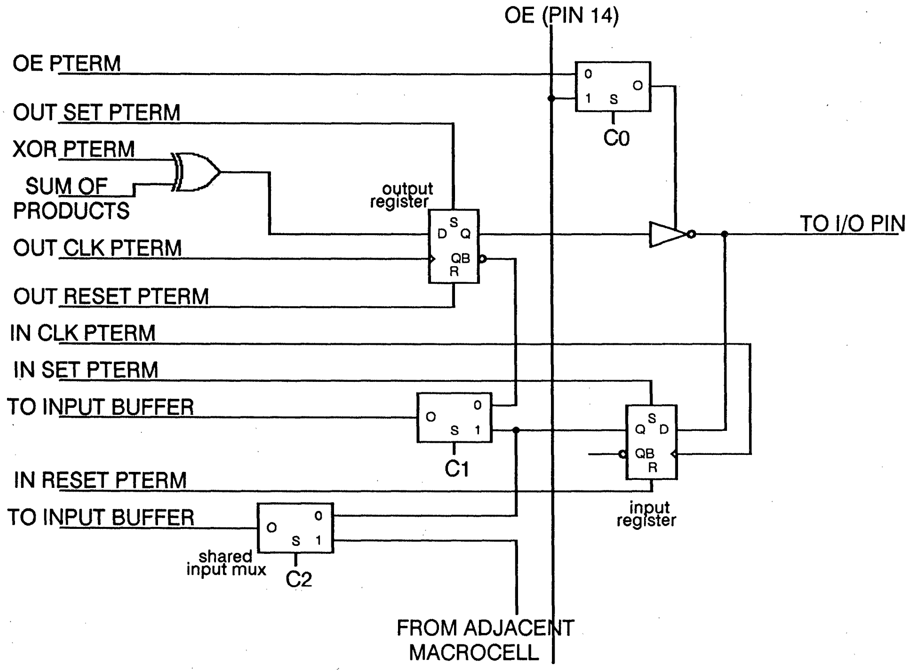


Figure 4. CY7C331 Macrocell

figured with a separate input and output flip-flop, which is highly useful in bus oriented applications. Each flip-flop has a separate product term for the clock, preset and reset. The D input of the output flip-flop incorporates an XOR with the sum-of-products array to allow selectable polarity, or implementation of a toggle or JK flip-flop. A unique feature of the macrocell flip-flops is the characteristic that when the set and reset inputs are both asserted, the flip-flop becomes transparent and the Q output follows the D input. Thus, the flip-flop can be used as a clocked register with an independent clock, set and reset, or as a combinational path. In addition, 6 shared input multiplexers are included in the CY7C331 which allow the user to bury up to 6 output flip-flops without giving up the input pins. The logic diagram of the I/O macrocell is illustrated in *Figure 4* and a block diagram of the CY7C331 is provided in *Figure 5.*

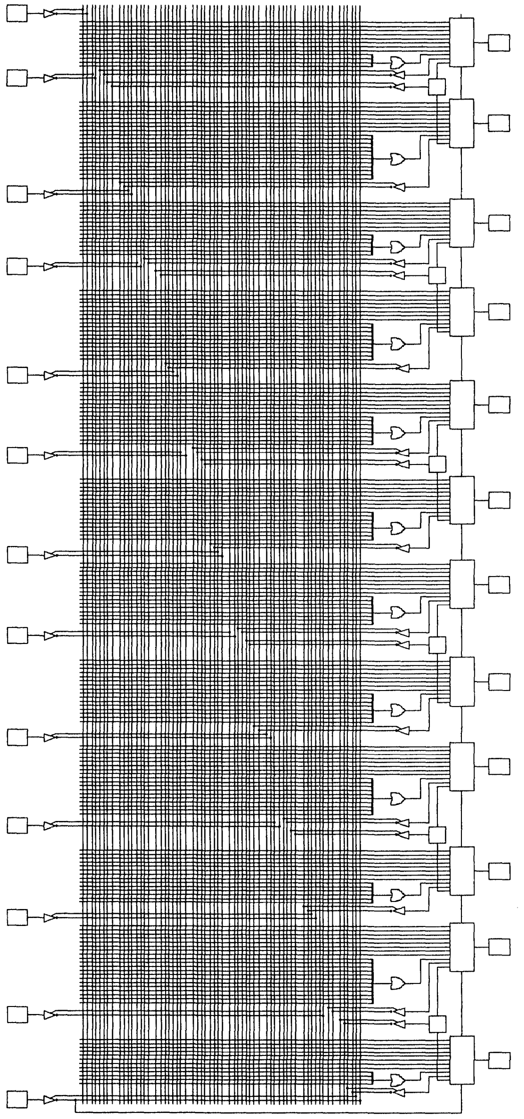


Figure 5. CY7C331 Block Diagram

4 Channel Interrupt Controller Design

A functional block diagram of the interrupt controller is provided in *Figure 2.* Major functional blocks include the Mask Register and Gating block, the Priority Encoder and Latch, and the Acknowledge Generator block. Pin assignments for the first-stage interrupt controller are defined in *Figure 6.*

Data Bus Interface

The data bus interface requires bidirectional operation. When CS and WE are asserted low, data is written into the mask register. When CS is asserted low and WE remains high, the current priority vector and interrupt status is held and is placed on the data bus. The I/O macrocell of the CY7C331 is readily adapted to the requirement. A logic diagram of the mask/priority vector function is illustrated in *Figure 7.* The interrupt status generation requires a different implementation. When a read cycle is detected (CS low, WE high), if any interrupt requests are currently pending, the interrupt status

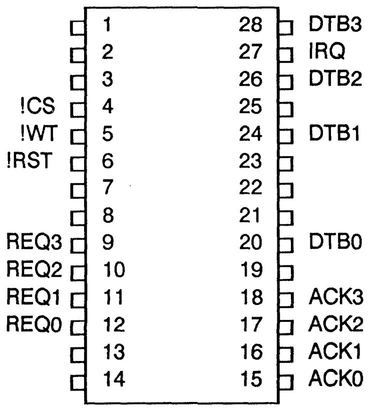


Figure 6. Interrupt Controller Pin Assignments

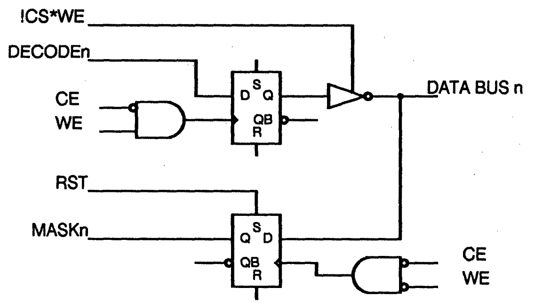


Figure 7. Mask/Priority Vector Function

bit must be asserted high. Furthermore, new interrupt requests are held off until the end of the read cycle. This requires a clocked implementation of the interrupt status bit on the data bus. This is illustrated in *Figure 8.* Note that the flip-flop is configured to be reset when CS is deasserted high.

Acknowledge Generation

Acknowledge generation requires that the priority vector being placed on the data bus be decoded and the corresponding Acknowledge line asserted until the Interrupt Request line is deasserted. There is a timing issue that must be resolved for proper operation. A valid priority vector is not available until after CS is asserted low. Thus, the proper channel cannot be decoded until the priority vector register has settled. A delay is required before the Acknowledge generation can be initiated. This can be accomplished in the following manner. The interrupt status bit is always asserted if there is a pending interrupt request and it occurs one propagation delay after CS is asserted on a read cycle. The Interrupt Status signal is then passed through an internal strobe stage which causes and an additional propagation delay. The Internal Strobe is then used to initiate the Acknowledge Generation sequence. The delayed strobe assures that the priority vector value has settled and the setup requirements for decoding have been met. The actual Acknowledge Generation function for a channel is implemented as a SR flip-flop which is set when a read cycle occurs, the priority vector corresponds to the channel and the delayed Internal Strobe occurs. The flip-flop is reset when the interrupt request for the channel is deasserted. A logic diagram for the Internal Strobe generation and a single Acknowledge Generation block is provided in *Figure 9* with a timing diagram illustrating typical operation in *Figure 10.*

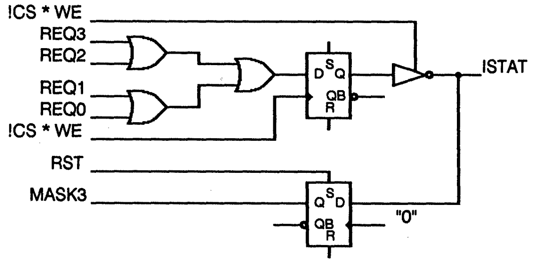


Figure 8. Interrupt Status Generation

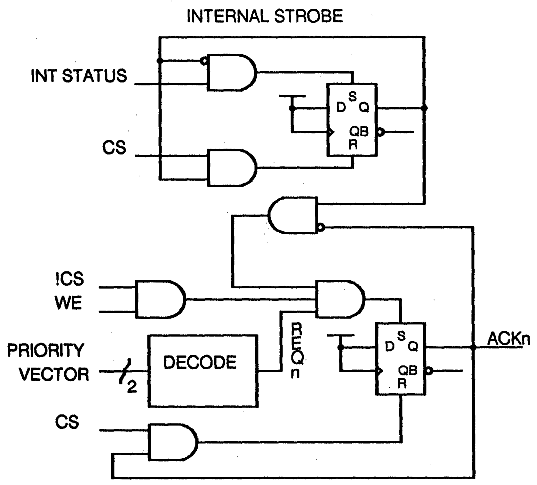


Figure 9. Internal Strobe/Acknowledge Generation

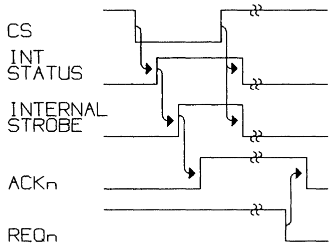


Figure 10. Timing Diagram

Logic Equations

The boolean equations for the interrupt controller are implemented using the syntax of the Cypress PLD Toolkit, which is a simple PLD assembler as shown below in *Appendix A*. The equations are heavily commented for clarity. The PLD Toolkit does not currently support "de-morganization" and because the CY7C331 contains inverting output buffers, boolean equations for output flip-flops are written for negative logic (i.e., solving for zero). In addition, the inversion requires swapping of the the preset and reset functions on the output flip-flops. Thus, the logical boolean equation required to "set" the flip-flop must be implemented on the "reset" of the flip-flop and in a similar manner the equation required to "reset" the flip-flop must be implemented on the "set" of the flip-flop.

Adding Cascade Capability

The interrupt controller design can be readily extended to accomodate four additional channels. The channels can be added by incorporating a cascade mechanism to allow a second interrupt controller to be attached to the first. The cascade method is illustrated in *Figure 11.* The additional channels require the format of the mask register and the interrupt vector to be extended. This extension is defined in *Figure 11.* The lower interrupt controller provides support for the lower priority interrupt channels, generates the IRQ to the processor and places the interrupt status and priority vector on the data bus during a read cycle. The upper interrupt controller supports the higher-priority channels and passes its current status and priority vector down to the lower interrupt controller. The interrupt status line is asserted high when the upper interrupt controller has a non-masked interrupt request pending. The upper interrupt controller is attached to the upper four bits of the data bus to allow the host processor to write into its mask register. However, because the upper interrupt controller passes its priority vector directly to the lower interrupt controller, there is no requirement for the upper interrupt controller to output any data on the bus during a read cycle. Operation of the cascaded version requires the lower interrupt controller to monitor the status interrupt line from the upper controller and incorporate it into the IRQ to the host processor and the interrupt vector placed on the data bus during a read cycle. Modification of the interrupt vector is straight forward. The upper interrupt channels have higher priority, so when the interrupt status from the upper controller is asserted, the lower 2 bits of the interrupt vector are the 2 vector bits from the upper controller. When the status is not asserted, the lower 2 bits of the interrupt vector are the lower priority interrupt vector encoded from the lower interrupt controller. The third bit of the interrupt vector is simply the state of the interrupt status signal from the upper controller. The modified interrupt controller equations for the lower element are shown in *Appendix B*. The upper element equations are shown in *Appendix C.*

Summary

This application note has offered a brief introduction to the CY7C331 asynchronous PLD and illustrated its flexibility in bus-oriented applications. The interrupt controller described is intended to serve as the basis for the design of flexible low-to-moderate complexity interrupt controllers. The design can be extended as required for different request polarity levels, edge sensitive inputs, or additional channels. A disk containing the PLD source files are available on request from the local Cypress Sales Office.

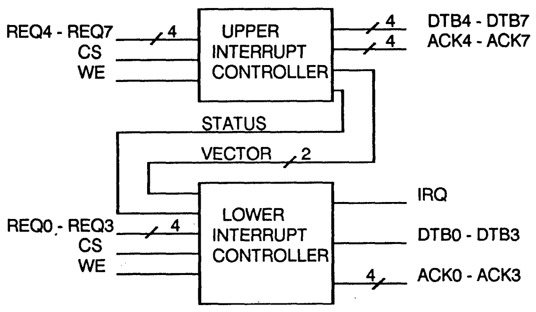


Figure 11. Block Diagram for Cascading Controllers

Appendix A. PLD ToolKit Source Code
Stand Alone Interrupt Controller

```
                                        {Stand Alone Interrupt Controller}
CY7C331;                                {declare device type}

CONFIGURE;

CS (node = 4),                          {pin 4, chip select}
WE (node = 5),                          {pin 5, write enable}
RST (node = 6),                         {pin 6, reset}
REQ3 (node = 9),                        {pin 9, interrupt request channel 3}
REQ2 (node = 10),                       {pin 10, interrupt request channel 2}
REQ1 (node = 11),                       {pin 11, interrupt request channel 1}
REQ0 (node = 12),                       {pin 12, interrupt request channel 0}

!IRQ (node = 27),                       {pin 27, interrupt to processor}
ISTAT (node = 28),                      {pin 28, data bus 3 - interrupt status}
PVEC2 (node = 26),                      {pin 26, data bus 2 - priority vector bit 2}
PVEC1 (node = 24),                      {pin 24, data bus 1 - priority vector bit 1}
PVEC0 (node = 20),                      {pin 20, data bus 0 - priority vector bit 0}
ACK3 (node = 18),                       {pin 18, acknowledge channel 3}
ACK2 (node = 17),                       {pin 17, acknowledge channel 2}
ACK1 (node = 16),                       {pin 16, acknowledge channel 1}
ACK0 (node = 15),                       {pin 15, acknowledge channel 0}
MSK3 (node = 34, SRC = 28),             {shared input mux for pin 28}
MSK2 (node = 33, SRC = 26),             {shared input mux for pin 26}
MSK1 (node = 32, SRC = 24),             {shared input mux for pin 24}
MSK0 (node = 31, SRC = 20),             {shared input mux for pin 20}

ISTB (node = 25),                       {pin 25, internal strobe}

EQUATIONS;

IRQ =   <oe>                            {no expression means always asserted, thus IRQ is always enabled.}
        <set_out>                       {make FF transparent}
        <clr_out>                       {make FF transparent}
        <xsum>                          {force invert}
        <sum> REQ3 & !ACK3 & !MSK3
        # REQ2 & !ACK2 & !MSK2
        # REQ1 & !ACK1 & !MSK1
        # REQ0 & !ACK0 & !MSK0;

!ISTAT =        <oe>!CS & WE
                <xsum>                  {force invert}
                <set_out> CS & ISTAT    {FF output is reset }
                <ck_out>!CS & WE
                <set_in>!RST            {interrupt is masked on reset}
                <ck_in>!WE & !CS
                <sum> REQ3 & !ACK3 & !MSK3
                # REQ2 & !ACK2 & !MSK2
                # REQ1 & !ACK1 & !MSK1
                # REQ0 & !ACK0 & !MSK0;

!PVEC2 =        <oe>!CS & WE
                <set_out>               {set is always asserted, thus pin is always zero}
                <set_in>!RST            {interrupt is masked on reset}
                <ck_in>!WE & !CS;
```

Appendix A. PLD ToolKit Source Code
Stand Alone Interrupt Controller (continued)

```
!PVEC1 =     <oe> !CS & WE
             <xsum>                                   {force invert}
             <ck_out> !CS & WE
             <sum> !ACK3 & REQ3 & !MSK3
             # !ACK2 & REQ2 & !MSK2
             <set_in>!RST                             {interrupt is masked on reset}
             <ck_in>!WE & !CS;

!PVEC0 =     <oe>!CS & WE
             <xsum>                                   {force invert}
             <ck_out>!CS & WE
             <sum> !ACK3 & REQ3 & !MSK3
                   # !ACK1 & REQ1 & !MSK1 & MSK2
                   # !MSK1 & !ACK1 & REQ1 & !REQ2
             <set_in>!RST                             {interrupt is masked on reset}
             <ck_in>!WE & !CS;

!ACK3 =      <oe>
             <clr_out>!CS & WE & PVEC1 &              {FF output is set }
                          PVEC0 & ISTB & !ACK3
             <set_out>CS & ACK3 & !REQ3;              {FF output is reset }

!ACK2 =      <oe>
             <clr_out>!CS & WE & PVEC1 &              {FF output is set}
                   !PVEC0 & ISTB & !ACK2
             <set_out>CS & ACK2 & !REQ2;              {FF output is reset }

!ACK1 =      <oe>
             <clr_out>!CS & WE & !PVEC1 &             {FF output is set}
                   PVEC0 & ISTB & !ACK1
             <set_out>CS & ACK1 & !REQ1;              {FF output is reset }

!ACK0 =      <oe>
             <clr_out> !CS & WE & !PVEC1 &            {FF output is set}
                   !PVEC0 & ISTB & !ACK0
             <set_out>CS & ACK0 & !REQ0;              {FF output is reset }

!ISTB =      <oe>
             <clr_out>ISTAT & !ISTB                   {FF output is set }
             <set_out>CS & ISTB;                      {FF output is reset }

                                                      {end of file}
```

Appendix B. PLD ToolKit Source Code
Cascadable Interrupt Controller-Lower Element

```
                                         {Cascaded Interrupt Controller - Lower Element}
CY7C331;                                 {declare device type}

CONFIGURE;
USTAT (node = 1),                        {pin 1, upper element interrupt status}
RVEC1 (node = 2),                        {pin 2, ripple vector bit 1 from upper element}
RVEC0 (node = 3),                        {pin 3, ripple vector bit 0 from upper element}
CS (node = 4),                           {pin 4, chip select}
WE (node = 5),                           {pin 5, write enable}
RST (node = 6),                          {pin 6, reset}
REQ3 (node = 9),                         {pin 9, interrupt request channel 3}
REQ2 (node = 10),                        {pin 10, interrupt request channel 2}
REQ1 (node = 11),                        {pin 11, interrupt request channel 1}
REQ0 (node = 12),                        {pin 12, interrupt request channel 0}

!IRQ (node = 27),                        {pin 27, interrupt to processor}
ISTAT (node = 28),                       {pin 28, data bus 3 - interrupt status}
PVEC2 (node = 26),                       {pin 26, data bus 2 - priority vector bit 2}
PVEC1 (node = 24),                       {pin 24, data bus 1 - priority vector bit 1}
PVEC0 (node = 20),                       {pin 20, data bus 0 - priority vector bit 0}
ACK3 (node = 18),                        {pin 18, acknowledge channel 3}
ACK2 (node = 17),                        {pin 17, acknowledge channel 2}
ACK1 (node = 16),                        {pin 16, acknowledge channel 1}
ACK0 (node = 15),                        {pin 15, acknowledge channel 0}
MSK3 (node = 34, SRC = 28),              {shared input mux for pin 28}
MSK2 (node = 33, SRC = 26),              {shared input mux for pin 26}
MSK1 (node = 32, SRC = 24),              {shared input mux for pin 24}
MSK0 (node = 31, SRC = 20),              {shared input mux for pin 20}

ISTB (node = 25),                        {pin 25, internal strobe}

EQUATIONS;

IRQ = <oe>
        <set_out>                        {make FF transparent}
        <clr_out>                        {make FF transparent}
        <xsum>                           {force invert}
        <sum> REQ3 & !ACK3 & !MSK3
        # REQ2 & !ACK2 & !MSK2
        # REQ1 & !ACK1 & !MSK1
        # REQ0 & !ACK0 & !MSK0
        # USTAT;

!ISTAT =    <oe> !CS & WE
            <xsum>                       {force invert}
            <set_out> CS & ISTAT         {FF output is reset }
            <ck_out> !CS & WE
            <set_in> !RST                {interrupt is masked on reset}
            <ck_in> !WE & !CS
            <sum> REQ3 & !ACK3 & !MSK3
            # REQ2 & !ACK2 & !MSK2
            # REQ1 & !ACK1 & !MSK1
            # REQ0 & !ACK0 & !MSK0
            # USTAT;
```

Appendix B. PLD ToolKit Source Code
Cascadable Interrupt Controller-Lower Element (continued)

```
!PVEC2 =    <oe> !CS & WE
            <xsum>                                  {force invert}
            <ck_out>!CS & WE
            <sum>USTAT
            <set_in>!RST                            {interrupt is masked on reset}
            <ck_in>!WE & !CS;

!PVEC1 =    <oe> !CS & WE
            <xsum>                                  {force invert}
            <ck_out> !CS & WE
            <sum> !ACK3 & REQ3 & !MSK3 & !USTAT
            # !ACK2 & REQ2 & !MSK2 & !USTAT
            # RVEC1 & USTAT
            <set_in>!RST                            {interrupt is masked on reset}
            <ck_in>!WE & !CS;

!PVEC0 =    <oe> !CS & WE
            <xsum>                                  {force invert}
            <ck_out>!CS & WE
            <sum> !ACK3 & REQ3 & !MSK3 & !USTAT
            # !ACK1 & REQ1 & !MSK1 & MSK2 & !USTAT
            # !MSK1 & !ACK1 & REQ1 & !REQ2 & !USTAT
            # RVEC0 & USTAT
            <set_in>!RST                            {interrupt is masked on reset}
            <ck_in>!WE & !CS;

!ACK3 =     <oe>
            <clr_out>!CS & WE & !PVEC2 & PVEC1 &    {FF output is set }
                    PVEC0 & ISTB & !ACK3
            <set_out>CS & ACK3 & !REQ3;             {FF output is reset }

!ACK2 =     <oe>
            <clr_out>!CS & WE & !PVEC2 &            {FF output is set }
                    PVEC1 & !PVEC0 & ISTB & !ACK2
            <set_out>CS & ACK2 & !REQ2;             {FF output is reset }

!ACK1 =     <oe>
            <clr_out>!CS & WE & !PVEC2 &            {FF output is set }
                    !PVEC1 & PVEC0 & ISTB & !ACK1
            <set_out>CS & ACK1 & !REQ1;             {FF output is reset }

!ACK0 =     <oe>
            <clr_out>!CS & WE & !PVEC2 &            {FF output is set }
                    !PVEC1 & !PVEC0 & ISTB & !ACK0
            <set_out>CS & ACK0 & !REQ0;             {FF output is reset }

 !ISTB =    <oe>
            <clr_out>ISTAT & !ISTB                  {FF output is set }
            <set_out>CS & ISTB;                     {FF output is reset }
                                 {end of file}
```

Appendix C. PLD ToolKit Source Code
Cascadable Interrupt Controller-Upper Element

```
CY7C331;                              {declare device type}
                                      {Cascaded Interrupt Controller - Upper Element}
CONFIGURE;

CS (node = 4),                        {pin 4, chip select}
WE (node = 5),                        {pin 5, write enable}
RST(node = 6),                        {pin 6, reset}
REQ3 (node = 9),                      {pin 9, interrupt request channel 3}
REQ2 (node = 10),                     {pin 10, interrupt request channel 2}
REQ1 (node = 11),                     {pin 11, interrupt request channel 1}
REQ0 (node = 12),                     {pin 12, interrupt request channel 0}
PVEC3 (node = 28),                    {pin 28, data bus 3 - always zero}
PVEC2 (node = 26),                    {pin 26, data bus 2 - always zero}
PVEC1 (node = 24),                    {pin 24, data bus 1 - always zero}
PVEC0 (node = 20),                    {pin 20, data bus 0 - always zero}
ACK3 (node = 25),                     {pin 25, acknowledge channel 3}
ACK2 (node = 23),                     {pin 23, acknowledge channel 2}
ACK1 (node = 19),                     {pin 19, acknowledge channel 1}
ACK0 (node = 17),                     {pin 17, acknowledge channel 0}
MSK3 (node = 34, SRC = 28),           {shared input mux for pin 28}
MSK2 (node = 33, SRC = 26),           {shared input mux for pin 26}
MSK1 (node = 32, SRC = 24),           {shared input mux for pin 24}
MSK0 (node = 31, SRC = 20),           {shared input mux for pin 20}
ISTB (node = 27),                     {pin 27, internal strobe}
USTAT(node = 18),                     {pin 18, interrupt status output}
ISENSE (node = 30, SRC = 18),         {shared input mux for pin 18}
                                      {internal interrupt sense to generate input for ISTB}
RVEC1 (node = 16),                    {pin 16, ripple vector bit 1 output}
RVEC0 (node = 15),                    {pin 15, ripple vector bit 0 output}

EQUATIONS;

!PVEC3 =    <set_out>                 {output always zero}
            <set_in>!RST              {interrupt is masked on reset}
            <ck_in>!WE & !CS;

!PVEC2 =    <set_out>                 {output always zero}
            <set_in>!RST              {interrupt is masked on reset}
            <ck_in>!WE & !CS;

!PVEC1 =    <xsum>                    {force invert}
            <ck_out>!CS & WE
            <sum> !ACK3 & REQ3 & !MSK3
            # !ACK2 & REQ2 & !MSK2
            <set_in>!RST              {interrupt is masked on reset}
            <ck_in>!WE & !CS;

!PVEC0 =    <xsum>                    {force invert}
            <ck_out> !CS & WE
            <sum> !ACK3 & REQ3 & !MSK3
            # !ACK1 & REQ1 & !MSK1 & MSK2
            # !MSK1 & !ACK1 & REQ1 & !REQ2
            <set_in>!RST              {interrupt is masked on reset}
            <ck_in>!WE & !CS;
```

Appendix C. PLD ToolKit Source Code
Cascadable Interrupt Controller-Upper Element (continued)

```
!ACK3 =      <oe>
             <clr_out>!CS & WE & PVEC1 &              {FF output is set }
                  PVEC0 & ISTB & !ACK3
             <set_out>CS & ACK3 & !REQ3;              {FF output is reset }

!ACK2 =      <oe>
             <clr_out>!CS & WE & PVEC1 &              {FF output is set }
                  !PVEC0 & ISTB & !ACK2
             <set_out>CS & ACK2 & !REQ2;              {FF output is reset }

!ACK1 =      <oe>
             <clr_out>!CS & WE & !PVEC1 &             {FF output is set }
                  PVEC0 & ISTB & !ACK1
             <set_out>CS & ACK1 & !REQ1;              {FF output is reset }

!ACK0 =      <oe>
             <clr_out>!CS & WE & !PVEC1 &             {FF output is set }
                  !PVEC0 & ISTB & !ACK0
             <set_out>CS & ACK0 & !REQ0;              {FF output is reset }

!USTAT =     <oe>
             <xsum>                                   {force invert}
             <set_out>                                {make FF transparent}
             <clr_out>                                {make FF transparent}
             <sum> REQ3 & !ACK3 & !MSK3
                  # REQ2 & !ACK2 & !MSK2
                  # REQ1 & !ACK1 & !MSK1
                  # REQ0 & !ACK0 & !MSK0
             <ck_in>!CS & WE
             <clr_in>CS & ISENSE;

!RVEC1 =     <oe>
             <xsum>                                   {force invert}
             <set_out>                                {make FF transparent}
             <clr_out>                                {make FF transparent}
             <sum>  !ACK3 & REQ3 & !MSK3
                  #  !ACK2 & REQ2 & !MSK2;

!RVEC0 =     <oe>
             <xsum>                                   {force invert}
             <set_out>                                {make FF transparent}
             <clr_out>                                {make FF transparent}
             <ck_out>!CS & WE
             <sum> !ACK3 & REQ3 & !MSK3
                  # !ACK1 & REQ1 & !MSK1 & MSK2
                  # !ACK1 & REQ1 & !MSK1 & !REQ2;

!ISTB =      <oe>
             <clr_out>ISENSE & !ISTB                  {FF output is set }
             <set_out>CS & ISTB;                      {FF output is reset }
                                          {end of file}
```

NOTES:

CYPRESS SEMICONDUCTOR

Using the CY7C331 as a Waveform Generator

Introduction

This application note demonstrates the capability of the Cypress CY7C331 CMOS Erasable Programmable Logic Device (EPLD) to support a design requiring multiple clocks, input registers, buried registers, and independent control of individual register's set and reset inputs. Combining this flexibility of design with high-speed performance has previously not been possible. The application example shown demonstrates the use of the CY7C331 as a programmable waveform generator.

The CY7C331 is a member of the Cypress slimline 28-pin family of high performance CMOS EPLDs. Family members are characterized by high speed, increased I/O, and high integration. The CY7C331 has a highly flexible architecture intended to support asynchronous and general purpose "glue" logic integration applications. The device has a 192 product term array and twelve I/O logic macrocells. Each macrocell has two D-type flip-flops with asynchronous set, reset, and bypass capability. The clock, preset, and reset inputs of a flip-flop are individually programmable. Output enable control and feedback are also individually programmable in each macrocell. Combinatorial and registered inputs, as well as buried states, are all easily supported by the CY7C331.

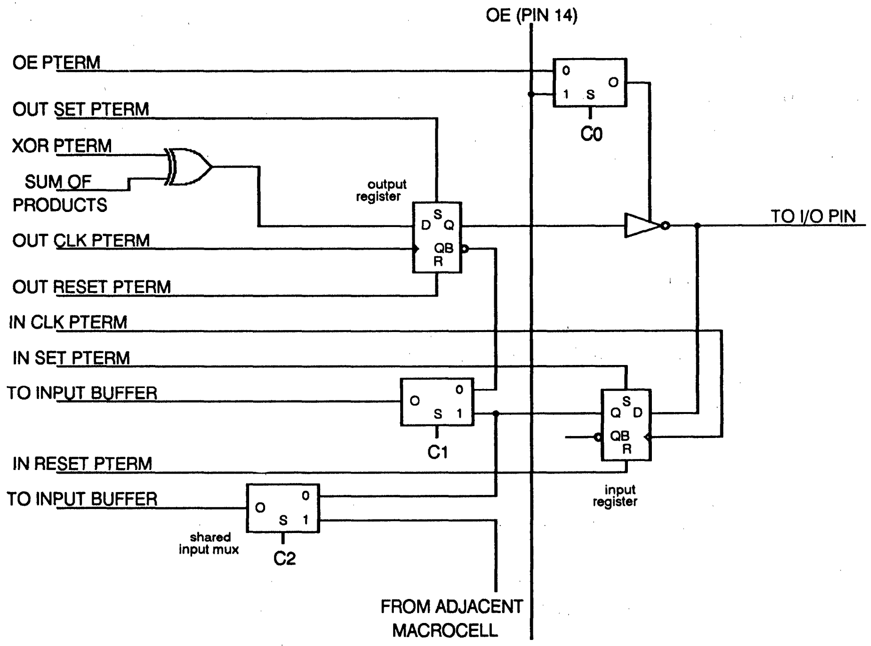


Figure 1. The CY7C331 Macrocell

The ability to bury registers and associated gates is highly desireable as it aids in increasing the number of "usable gates" in an EPLD. Typically, if an I/O pin is used as an input, the corresponding output register and its supporting product term structure is wasted. This loss occurs because only one macrocell feedback path is present. When this path is used by the I/O pin (as an input) no register feedback path is available, and the contents of the register cannot be fed back into the array.

The dual muxing structure of the CY7C331 prevents this limitation by allowing the designer to make use of the shared input mux (see *Figure 1*) as an I/O path into the array, while simultaneously feeding back the registered contents using the separate macrocell feedback mux. Because the output register can be made to be transparent by asserting both the register's set and preset nodes, simultaneous combinatorial feedback can also be achieved. Use of this feature allows the implementation of bidirectional I/O in both registered and combinatorial configurations.

Figure 2 contains PLD ToolKit source code that configures an I/O macrocell as bidirectional, with feedback from the output. The I/O pin corresponding the macrocell will be labeled IO_PIN and each line of code is commented to explain what it accomplishes.

Note that IO_PIN is assigned to node 28, and IN_PATH is assigned to node 34, with pin 28 as a source. In the simulator, the input waveform must be added on the trace corresponding to node 28, even though that trace is named IO_PIN. IN_PATH will be assigned to node 34, which is a read only node. This is true even if IO_PIN is configured as a buried register, and IN_PATH is always an input. The reason for this is that node 34 is just a mux, and the register associated with the input belongs to node (pin) 28. If you wish to see the value of the output register when the pin is an input, you can create a view node for the node. This allows the user to probe several different places inside a macrocell. For more information on view nodes, consult the PLD ToolKit Manual, Chapter 4.3.

```
{*************************************************************************************}
CY7C331;       {The first line of code selects the device }
CONFIGURE;  {In this section pin and node names are specified, along with configuration information}

INCLK, OUTCLK, /INCLR, /INSET, OE1, /OE2, INPUT, /OUTCLR(NODE=9), /OUTSET,
{The input names are listed above. Pin 1 will be the input clock, pin 2 will be the output clock. Pins 3 and 4
will be the input register's clear and set signals respectively. Pins 5 and 6 will be output enables, OE1 is high
asserted, /OE2 is low asserted. Pin 7 is a straight input. We skip pin 8 because it is Vss. Pins 9 and 10 will
be the input register's clear and set signals.}
IO_PIN(NODE=28, IREG), IN_PATH(NODE=34, SRC=28), OUT(NODE=27),
{Pin 28 is the actual bidirectional pin. The IREG attribute specifies that the input to the array comes from
the output register, rather than the pin. Node 34 is the shared input mux for nodes 27 and 28. IN_PATH is
the input path to the array from pin 28. Pin 27 is a simple output.}
EQUATIONS;  {This is where the array is specified.}

IO_PIN =       <SUM> INPUT         {When IO_PIN is an output, it follows Pin 7.}
               <SET_OUT> OUTSET
               <CLR_OUT> OUTCLR
               <CLK_OUT> OUTCLK
               <OE> OE1 * OE2      {Outputs are enabled when OE_1 is high, and /OE_2 is low.}
               <CLK_IN> INCLK
               <CLR_IN> INCLR
               <SET_IN> INSET;

OUT =          <OE>           {Listing the connective alone sets the product term to "1", always asserted.}
               <SET_OUT>  {When both the set and reset product terms are asserted, the register }
               <CLR_OUT>  {becomes transparent. Thus, this is a combinatorial output.}
               <SUM> IN_PATH;  {This output always shows the value of the input register at pin 28.}
                              {If the register is in combinatorial mode, the value on pin 28 will be shown.}
```

Figure 2. PLD ToolKit Source Code for Bidirectional Pin

The CY7C331 as a Function Generator

Waveform generators are useful in a variety of a applications, primarily in the test and diagnostic areas. Any time high-speed digital waveforms must be created, a programmable waveform generator is the ideal solution. This CY7C331 solution allows waveforms of frequencies greater than 30 MHz to be generated.

This waveform generator builds waveforms with respect to a system clock called SYS_CLK. The number of cycles of SYS_CLK that the output waveform (OUT_WAVE) should be low is loaded into LOW_REG(2:0). The number of cycles of SYS_CLK that OUT_WAVE should be high is loaded into HI_REG(2:0). For this implementation, these values must be between 2 and 7.

When the START signal is asserted, OUT_WAVE goes low, and LOW_REG(2:0) is loaded into a counter. When the count is almost 0, the signal TERM_CNT is deasserted, then reasserted when the count reaches 0. This toggles OUT_WAVE, and loads a second counter with the value in HI_REG(2:0). The cycle repeats, alternating between HI_REG(2:0) and LOW_REG(2:0) until SYS_CLK is witheld, or new values are loaded into HI_REG(2:0) and LOW_REG(2:0), and START is reissued. *Figure 3* depicts the waveforms for this design.

HI_REG(2:0) and LOW_REG(2:0) are loaded using /DSTRB and ADDR(7:0). The user can specify any address for these registers. In this example, HI_REG(2:0) is at ADDR(7:0) = 00 Hex, and LOW_REG(2:0) is at ADDR(7:0) = 01 Hex.

The implementation of this design requires two separate three bit input registers, decoding logic for the input register clocks, two separate three bit counters, logic and two miscellaneous registers. In this design, all the counter flip-flops must be individually settable or resettable. In addition, there are four separate clocking functions.

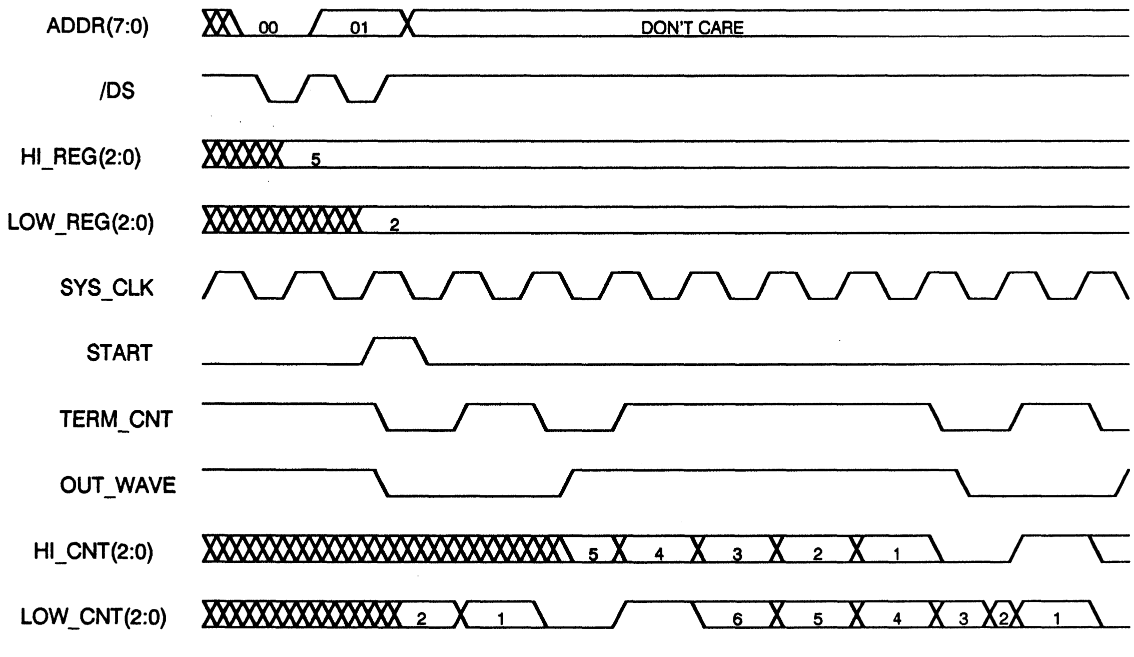


Figure 3. Waveform Generator Internal/External Waveforms

This type of design is historically very difficult to implement in a PLD. Typically the use of the preset and reset inputs on individual flip-flops is not available nor is separate clocking of those flip-flops. Because the CY7C331 has these features, implementation of this design was effortless.

Figure 4 shows the SSI implementation of this design. LOW_IN_CLK is the clock input for LOW_REG(2:0). It is the result of decoding the active low /DS (data strobe) and ADDR(7:0) = 01 Hex. HI_IN_CLK, is similarly decoded with /DS and ADDR(7:0) = 00 Hex.

LOW_CNT_(2:0) and HI_CNT_(2:0) form 2 three bit counters. These counters are loaded with the contents of the LOW_REG(2:0) and HI_REG(2:0) registers respectively, by using the individual set and reset on each flip-flop. LOW_CNT_(2:0) is loaded when /TERM_CNT is low and OUT_WAVE is high. Similarly, HI_CNT_(2:0) is loaded when /TERM_CNT is low and OUT_WAVE is low. The counters are both clocked with SYS_CLK.

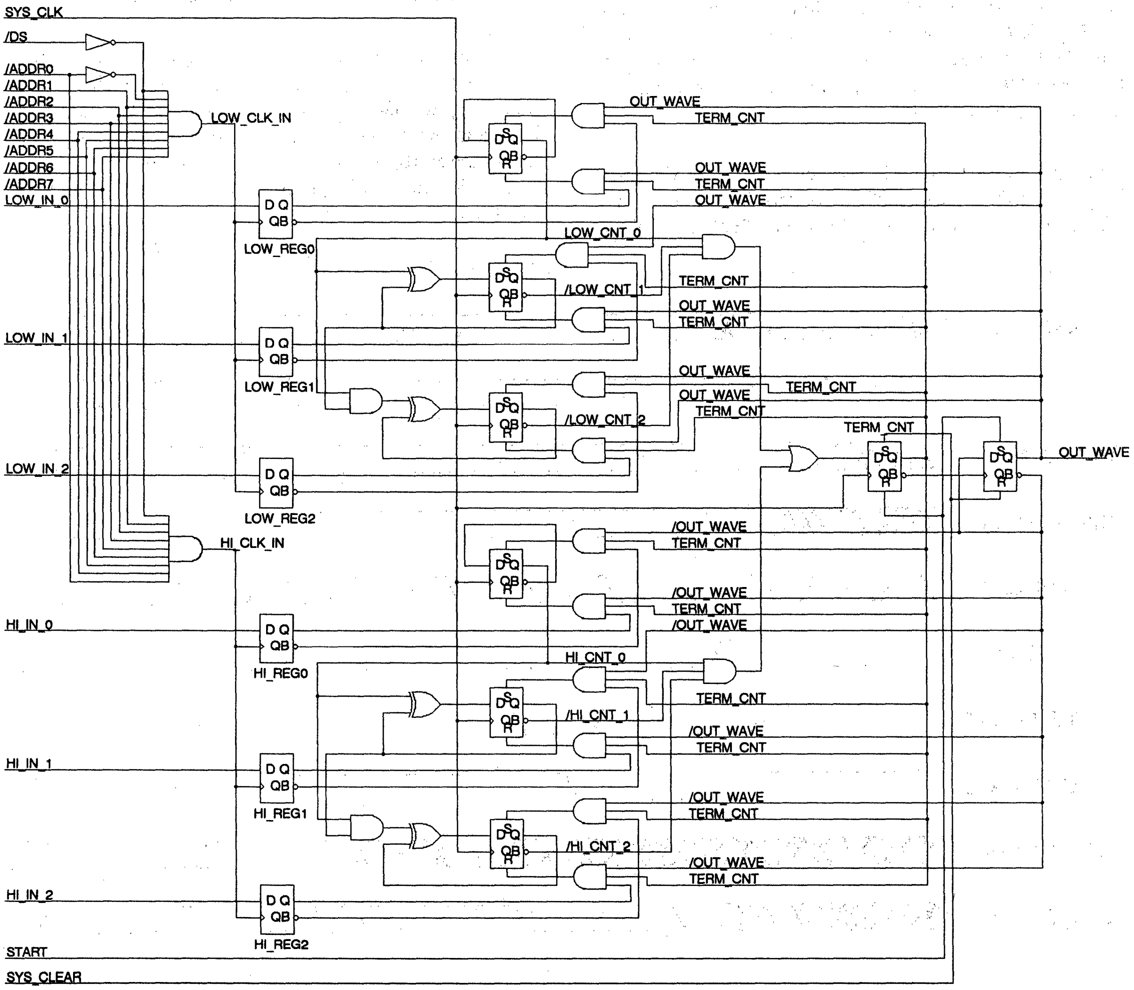


Figure 4. Waveform Generator Schematic

/TERM_CNT is also clocked by SYS_CLK and it detects when either of the counters are equal to 1. When a counter reaches 1, /TERM_CNT goes low for one clock, and then goes high again. The rising edge of /TERM_CNT is used to clock OUT_WAVE, which is configured to toggle on every clock.

PLD ToolKit Implementation

Appendix A contains the Cypress PLD ToolKit implementation of the waveform generator discussed in this application note. There are two areas which may require some clarification, These are the pin assignments and polarity.

The pin assignments for nodes (pins) 1 through 14 are straightforward. Pin 8 has been skipped because it is a Vss pin. These pins are the combinatorial inputs of the CY7C331, so no configuration information is needed.

OUT_WAVE is assigned to pin 16. "IOP" following the node assignment indicates that the feedback mux is programmed to feed back the Q output of the OUT_WAVE register. This is actually the default so it does not need to be specified. It has been included here for documentation purposes. The same is true for TERM_CNT, /HI_CNT_0 and /LOW_CNT_1. Notice that HI_IN_1 and LOW_IN_0 have the attribute "IREG" listed after the node assignment. This specifies that these pins are dedicated inputs, that is the feedback mux is configured to select the Q output of the input register associated with the pin, as opposed to the Q output of the output register. This is an override of the default discussed above.

The rest of the assignments are of the same form as /HI_CNT_2 and HI_IN_2. /HI_CNT_2 is assigned to node 18, with an attribute of IOP. As mentioned earlier, this configures the feedback mux to select feedback from /HI_CNT_2 as the array input. HI_IN_2 has been assigned to node 30, with "SRC=18". Node 30 is a shared input mux that serves as an input path from either the input register on pin 18 or the input register on pin 17. SRC=18 specifies that HI_IN_2 is assigned to the input register on pin 18. (The default is that the even pin is always selected. Again the statement "SRC=18" has been included primarily for documentation purposes. This method for utilizing both the input and output registers of a pin is used 4 times in this design. In all of these cases, the output register is buried (not accessible to the pin). *Figure 5* is a footprint of the CY7C331 with all external pin signals labeled.

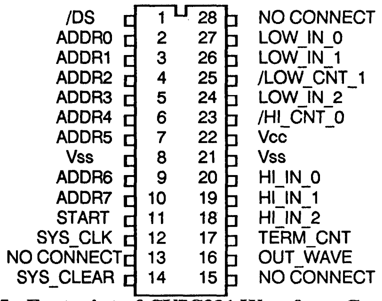


Figure 5. Footprint of CY7C331 Waveform Generator

A close look at the file in *Appendix A* may also raise some questions concerning polarity conventions in the PLD ToolKit. Polarity on inputs is fairly straightforward. Note that the "/" in /START means that this is a low asserted signal. When START appears in the EQUATIONS section (refer to /OUT_WAVE and /TERM_CNT equations) this is interpreted as /START being asserted. Thus, when /START=0, the OUT_WAVE register is set.

This leads us to the more confusing case of output feedback polarity. Polarity on the CY7C331 is not programmable, unless it is done using the XOR in the array. Thus when TERM_CNT is specified in the CONFIGURATION section, this means that the output register is /TERM_CNT because there is an inversion between the register output and the pin. This means that when you set TERM_CNT, the pin will be low. How, then, do you specify that TERM_CNT is asserted when it appears on the right of an equation? The answer is that you refer to the polarity present on the pin. Thus, in the <CK_OUT> portion of the equation for /OUT_WAVE, is is specified TERM_CNT. This means that /OUT_WAVE is clocked when pin 17 (TERM_CNT) exhibits a rising edge.

Appendix A. PLD Toolkit Source Code for the Waveform Generator

```
CY7C331;

CONFIGURE;
/DS,                                              {Low asserted data strobe}
ADDR0, ADDR1, ADDR2, ADDR3, ADDR4, ADDR5,         {address bits 0,1,2,3,4,5,}
ADDR6(NODE=9), ADDR7                              {address bits 6 and 7}
/START,                                           {start sequence}
SYS_CLK,                                          {counter clock}
SYS_CLEAR(NODE=14),                               {initialize OUT_WAVE,TERM_CNT to a quiescent state}
OUT_WAVE(NODE=16,IOP),                            {output wave form}
TERM_CNT(NODE=17,IOP),                            {terminal count decode register}
/HI_CNT_2(NODE=18,IOP),                           {high counter bit 2, a buried register}
HI_IN_2(NODE=30,SRC=18),                          {high register input bit 2}
HI_IN_1(NODE=19,IREG),                            {high counter input bit 1}
/HI_CNT_1(NODE=20,IOP),                           {high counter bit 1, a buried register}
HI_IN_0(NODE=31,SRC=20),                          {pin 20 acts as high register input bit 0}
/HI_CNT_0(NODE=23,IOP),                           {high counter bit 0}
/LOW_CNT_2(NODE=24,IOP),                          {low counter bit 2, a buried register}
LOW_IN_2(NODE=32,SRC=24),                         {pin 24 is low register input bit 2}
/LOW_CNT_1(NODE=25,IOP),                          {low counter bit 1}
LOW_IN_1(NODE=33,SRC=26),                         {pin 26 acts as low register input bit 1}
/LOW_CNT_0(NODE=26,IOP),                          {low counter bit 1, a buried register}
LOW_IN_0(NODE=27,IREG),                           {low register input bit 0}

EQUATIONS;
 LOW_CNT_0 :=      <SUM> /LOW_CNT_0
                   <CK_OUT> SYS_CLK
                   <CK_IN> DS*ADDR0*/ADDR1*/ADDR2*/ADDR3*/ADDR4*/ADDR5*/ADDR6*/ADDR7
                   <SET_OUT> /LOW_IN_0 * /OUT_WAVE * /TERM_CNT
                   <CLR_OUT> LOW_IN_0 * /OUT_WAVE * /TERM_CNT;

/LOW_IN_0 =        <CK_IN> DS*ADDR0*/ADDR1*/ADDR2*/ADDR3*/ADDR4*/ADDR5*/ADDR6*/ADDR7;

LOW_CNT_1 :=       <SUM> LOW_CNT_1
                   <XSUM> LOW_CNT_0
                   <SET_OUT> /LOW_IN_1 * /OUT_WAVE * /TERM_CNT
                   <CLR_OUT> LOW_IN_1 * /OUT_WAVE * /TERM_CNT
                   <CK_OUT> SYS_CLK
                   <OE>;

LOW_CNT_2 :=       <SUM> LOW_CNT_2
                   <XSUM> LOW_CNT_0 * LOW_CNT_1
                   <SET_OUT> /LOW_IN_2 * /OUT_WAVE * /TERM_CNT
                   <CLR_OUT> LOW_IN_2 * /OUT_WAVE * /TERM_CNT
                   <CK_OUT> SYS_CLK
                   <CK_IN> DS*ADDR0*/ADDR1*/ADDR2*/ADDR3*/ADDR4*/ADDR5*/ADDR6*/ADDR7;

/OUT_WAVE :=       <SUM> OUT_WAVE
                   <CK_OUT> TERM_CNT
                   <SET_OUT> START
                   <CLR_OUT> SYS_CLEAR
                   <OE>;
```

Appendix A. PLD Toolkit Source Code for the Waveform Generator (continued)

```
/TERM_CNT :=        <SUM> /LOW_CNT_0 * LOW_CNT_1 * LOW_CNT_2
                    <SUM> /HI_CNT_0 * HI_CNT_1 * HI_CNT_2
                    <CK_OUT> SYS_CLK
                    <CLR_OUT> START
                    <SET_OUT> SYS_CLEAR
                    <OE>;

HI_CNT_0 :=   <SUM> /HI_CNT_0
              <CK_OUT> SYS_CLK
              <OE>
              <CLR_OUT> HI_IN_0 * OUT_WAVE * /TERM_CNT
              <SET_OUT> /HI_IN_0 * OUT_WAVE * /TERM_CNT;

HI_CNT_1 :=   <SUM> HI_CNT_1
              <XSUM> HI_CNT_0
              <SET_OUT> /HI_IN_1*OUT_WAVE*/TERM_CNT
              <CLR_OUT> HI_IN_1*OUT_WAVE*/TERM_CNT
              <CK_OUT> SYS_CLK
              <CK_IN> DS*/ADDR0*/ADDR1*/ADDR2*/ADDR3*/ADDR4*/ADDR5*/ADDR6*/ADDR7;

/HI_IN_1 =    <CK_IN> DS*/ADDR0*/ADDR1*/ADDR2*/ADDR3*/ADDR4*/ADDR5*/ADDR6*/ADDR7;

HI_CNT_2 :=   <SUM> HI_CNT_2
              <XSUM> HI_CNT_1*HI_CNT_0
              <SET_OUT> /HI_IN_2*OUT_WAVE*/TERM_CNT
              <CLR_OUT> HI_IN_2*OUT_WAVE*/TERM_CNT
              <CK_OUT> SYS_CLK
              <CK_IN> DS*/ADDR0*/ADDR1*/ADDR2*/ADDR3*/ADDR4*/ADDR5*/ADDR6*/ADDR7;
```

NOTES:

Section Contents

CYPRESS SEMICONDUCTOR

Microcoded Systems Performance

The microcoded processor family of devices offered by Cypress Semiconductor are the fastest available. High performance systems designed for specific applications can be configured using this high performance chip set. The performance of these devices in 16- and 32-bit processors is detailed below.

Increasing functional integration is evident in the CY7C9101 16-bit slice, which is the equivalent to four CY7C901s (4-bit slice) and a 2902 carry lookahead generator. By placing these functions on a single chip, the interconnect delays between chips are reduced. Significant improvement in overall system throughput, reduced board space, and reduced power requirements are among the advantages of the CY7C9101 systems over CY7C901 based systems. Following is a critical path timing analysis of the data loop and control loop for generic 16- and 32-bit systems. A discussion of the speed and power advantages offered by CY7C9101 systems will also be presented.

Minimum Cycle Time Calculations for 16- and 32-Bit Systems

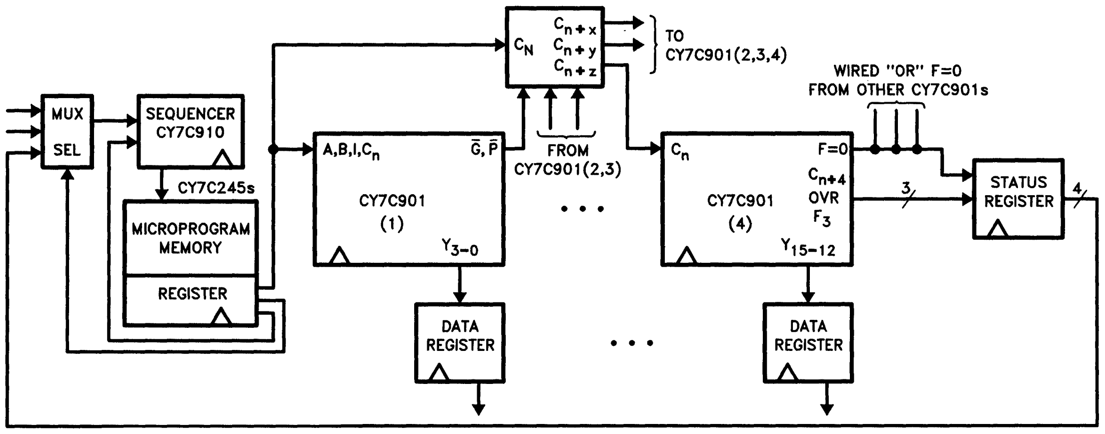


Figure 1. CY7C901 Based 16-Bit System (Pipelined System, Add without Simultaneous Shift)

	Data Loop	
CY7C245	Clock to Output	12
CY7C901	A, B to $\overline{G}$, $\overline{P}$	28
Carry Logic	$\overline{G}_0$, $\overline{P}_0$ to C_{n+z}	9
CY7C901	C_n to Worst Case	18
Register	Setup	4
		71 ns

	Control Loop	
CY7C245	Clock to Output	12
MUX	Select to Output	12
CY7C910	CC to Output	22
CY7C245	Access Time	20
		66 ns

Minimum Clock Period = 71 ns

October 1986

Minimum Cycle Time Calculations for 16- and 32-Bit Systems (Continued)

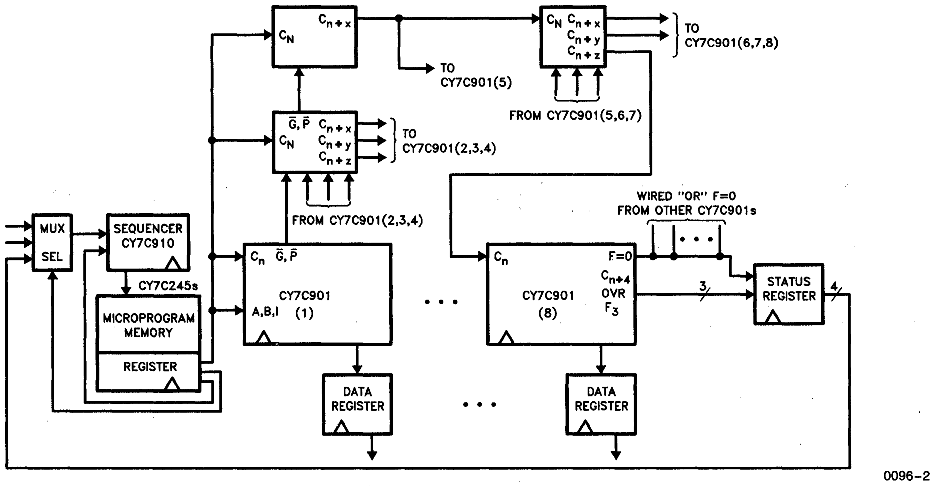


Figure 2. CY7C901 Based 32-Bit System (Pipelined System, Add without Simultaneous Shift)

Data Loop

CY7C245	Clock to Output	12
CY7C901	A, B to $\overline{G}$, $\overline{P}$	28
Carry Logic	$\overline{G}_0$, $\overline{P}_0$ to $\overline{G}$, $\overline{P}$	12
	$\overline{G}_0$, $\overline{P}_0$ to $C_n + x$	9
	C_n to $C_n + x, y, z$	14
CY7C901	C_n to Worst Case	18
Register	Setup	4
		97 ns

Control Loop

CY7C245	Clock to Output	12
MUX	Select to Output	12
CY7C910	CC to Output	22
CY7C245	Access Time	20
		66 ns

Minimum Clock Period = 97 ns

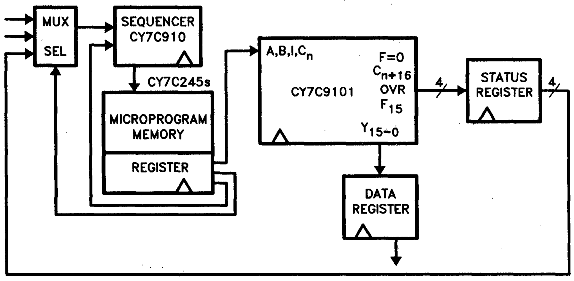


Figure 3. CY7C9101 Based 16-Bit System (Pipelined System, Add without Simultaneous Shift)

Data Loop

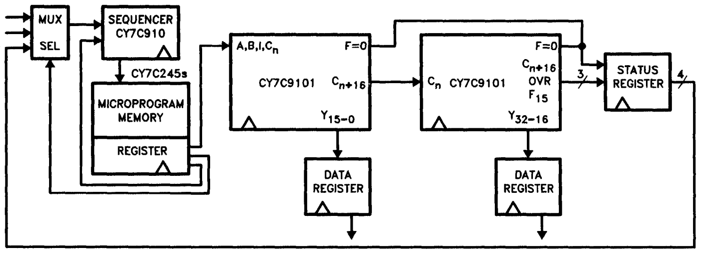
CY7C245	Clock to Output	12
CY7C9101	A, B to Y, C_{n+16}, OVR	37
Register	Setup	4
		53 ns

Control Loop

CY7C245	Clock to Output	12
MUX	Select to Output	12
CY7C910	CC to Output	22
CY7C245	Access Time	20
		66 ns

Minimum Clock Period = 66 ns

Minimum Cycle Time Calculations for 16- and 32-Bit Systems (Continued)



0096-4

Figure 4. CY7C9101 Based 32-Bit System (Pipelined System, Add without Simultaneous Shift)

	Data Loop	
CY7C245	Clock to Output	12
CY7C9101	A, B to $C_n + 16$	35
CY7C9101	C_n to Worst Case	24
Register	Setup	4
		75 ns

	Control Loop	
CY7C245	Clock to Output	12
MUX	Select to Output	12
CY7C910	CC to Output	22
CY7C245	Access Time	20
		66 ns

Minimum Clock Period = 75 ns

Power is an important consideration in microcoded systems. For an equivalent system, the CY7C901 offers substantial savings in power over the bipolar devices. Coupled with other low power Cypress CMOS devices, the power savings over bipolar is clearly evident. The functional integration of four CY7C901s with carry lookahead gives the CY7C9101 even greater advantages. The number of ALU elements is reduced by a factor of four, also, there is a reduction in the carry logic needed. A comparison between bipolar, CY7C901-based, and CY7C9101-based systems is given below in Table 1. Note that in this comparison, the devices common to all 16- and 32-bit system configurations are included in the I_{CC} computations.

Cypress CMOS devices offer the highest speed microcoded solutions while keeping power consumption to reasonable levels. The CY7C901-based systems win over bipolar's fastest devices in a speed comparison, while consuming roughly ⅓ the power. Upgrading to the CY7C9101 will result in even faster systems, at close to ⅓ the power of the CY7C901-based systems. This comparison is illustrated below, in Table 2.

Table 1

I_{CC} Calculations for 16-Bit Systems (All Figures in mA)

	Cypress CMOS		**Bipolar**
	CY7C901 Based	**CY7C9101 Based**	
Sequencer	100	100	340
Registered PROM	90	90	185
Carry Logic	110	—	110
ALU Elements			
4x Four-Bit Slice	320		1060
16-Bit Slice		75	
Total	620	265	1695

I_{CC} Calculations for 32-Bit Systems (All Figures in mA)

	Cypress CMOS		Bipolar
	CY7C901 Based	CY7C9101 Based	
Sequencer	100	100	340
Registered PROM	90	90	185
Carry Logic	330	110	330
ALU Elements			
8x Four-Bit Slice	640		2120
2x Sixteen-Bit Slice		150	
Total	1160	450	2975

Table 2. Speed/Power Comparison between Bipolar, CY7C901, CY7C9101

	Minimum Clock Cycle (ns)			**Maximum I_{CC} (mA)**		
	Bipolar	**CY7C901**	**CY7C9101**	**Bipolar**	**CY7C901**	**CY7C9101**
16-Bit Systems	85	71	66	1695	620	265
32-Bit Systems	111	97	75	2975	1160	450

NOTES:

CYPRESS SEMICONDUCTOR

Systems with CMOS 16-bit Microprogrammed ALUs

Introduction

In the past, the dominant use of microprogrammed Arithmetic and Logic Units (ALUs) has been as general-purpose data processors in computers. The reason for using microprogrammed machines in these applications was to improve performance, i.e., general purpose microprocessors were too slow. Microprogrammed processors, in addition to allowing custom instruction sets, were the only way to achieve the desired MIPS (Millions of Instructions Per Second) rate. However, with the advent of high performance, 20 MIPS Reduced Instruction Set Computers (RISC), microprogrammed ALUs have relinquished the general-purpose data processor application and moved to custom processors or special-purpose controllers. This application brief shows how to improve reliability, flexibility, and speed by diagramming timing and highlighting applications that benefit significantly from the

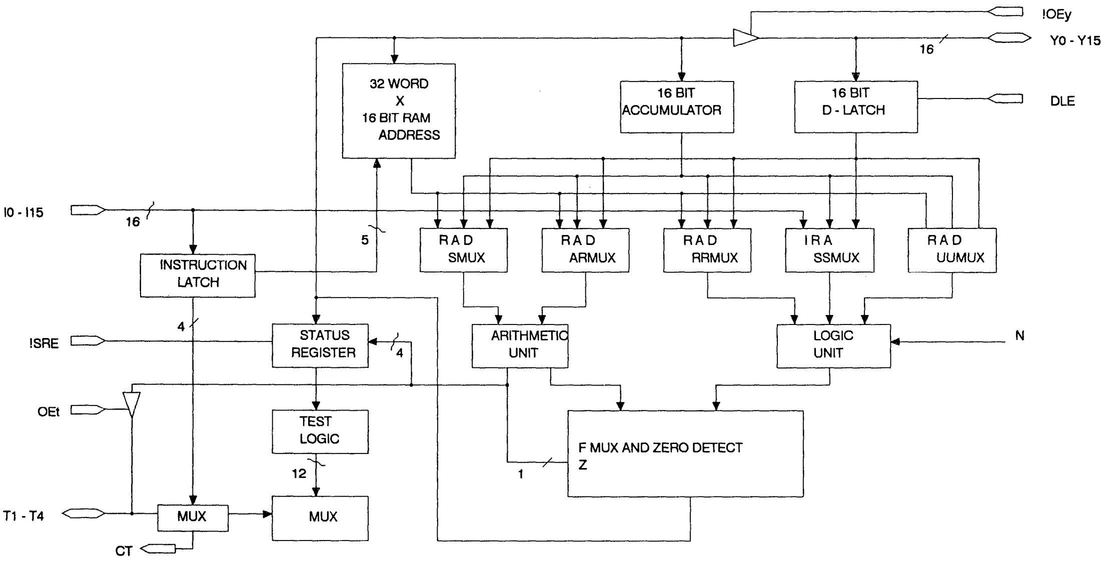


Figure 1. CY7C9116 Block Diagram

features of the 7C9116/7 architecture and CMOS technology.

7C9116/7 Architecture and Implementation

The 7C9116 and 7C9117 are extremely fast arithmetic and logic units implemented in a 1.2-micron double-metal CMOS process technology. As shown in *Figures 1* and *2*, the 7C9117 differs from the 7C9116 by incorporating a separate bus for data input (D) and for data output (Y) and thus allows for the design of faster microprogrammed systems. Both units are capable of worst-case propagation delays from instruction in to data out of 35 ns.

The 7C9116/7 contain single port 32 x 16 bit word register files, two operand arithmetic units, and three input logic units. Carry look ahead logic is also integrated together with the logic and arithmetic units. The instruction set of the 7C9116/7 can be divided into eleven types as listed in *Table 1*. Single clock operation is attained on the extensive bit manipulation and rotate instructions via the on-chip barrel shifter. In fact, all instructions in the ALU execute within one clock except for immediate instructions, where a second clock is needed to obtain the immediate operand.

Table 1. 7C9116/7 Instruction Types

Instruction Type		Example	
Single Operand	inc:	src plus 1-> dest	
Two Operand	add:	src plus src -> dest	
Single Bit Shift	shup1:	src up 1	
Bit Oriented		setnr:	set RAM bit n
Rotate by n bits		rotrl:	rotate RAM n bits
Rotate & Merge		mdai:	rotate src and src' w/mask
Rotate & Compare		rotc:	rotate src cmp w/src' set cc
Prioritize		prtnr:	indicate highest priority bit
CRC	crcf:	create crc fwd from qlink	
Status		rstst:	reset status register
No-Op		noop:	no effect

The 7C9116/7 are TTL-compatible and fully interchangeable with their counterparts from Advanced Micro Devices and Texas Instruments. However, caution should be exercised when illegal instructions or undefined opcodes are used. As the results are not predictable or guaranteed during these operations, they should not be used in any production system. *Table 2*

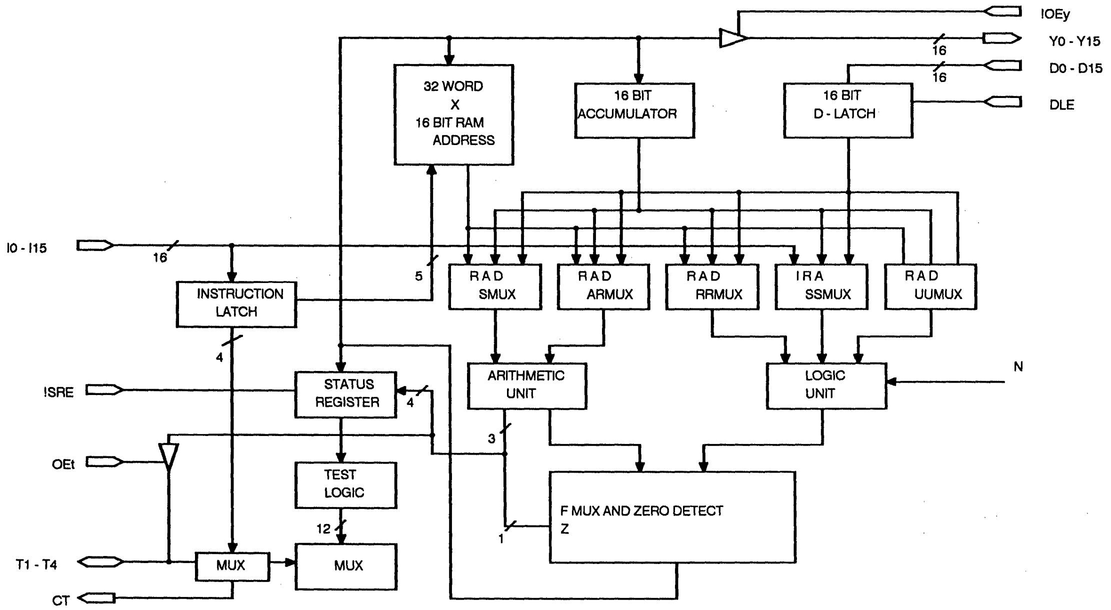


Figure 2. CY7C9117 Block Diagram

shows an example of such a condition, when SOA is mistakenly encoded as an undefined operation.

Table 2. Example Instruction Encoding Error

	SOA instruction:	ACC -> Y bus
Vendor	Instruction Code	Result
	Correct encoding:	
All	1111 1000 1000 0000	0000 1110 0000 1101
	Coding Error	
AMD	1110 0110 1000 0000	1111 1111 1111 1110
Cypress	1110 0110 1000 0000	1111 0100 1000 1100
TI	1110 0110 1000 0000	0000 0000 0000 1001

Another feature of the Cypress 7C9116/7 is that it allows the priority instruction to operate with both the source and destination as the accumulator. A secondary caution that could produce incorrect results is that older implementations of this architecture in bipolar technology do not allow such an operation. When using older bipolar implementations or testing devices, it should be noted that some machines may behave improperly and that undefined or illegal operations may produce different results for various device types depending on vendor and technology.

CMOS 16-Bit ALU - Faster Operation and Lower Power

Advanced microprogrammed architecture, combined with Cypress CMOS process technology, has multiple benefits to the design engineer. Custom computing units and controllers can operate at higher frequencies and consume less power, about 80% less, while being more reliable. *Table 3* compares the performance and power characteristics between a typical 16-bit microprogrammed ALU and the 7C9116/7. The results show that in addition to power savings, the 7C9116/7's reliability is enhanced by operating at lower die temperatures.

Other aspects of the 7C9116/7 CMOS processing technology also contribute to increased system reliability. In

Table 3. CMOS vs. Bipolar Performance and Power

	Cypress 7C9116/7	Generic 16-bit slice
Speed (ns)	35	53
Power (Icc, mA)		
Stactic	30	400
Max @10Mhz	150	600
Technology	CMOS	Bipolar

the past, CMOS technologies experienced problems with destructive latch-up conditions. Cypress CMOS processes minimize this problem by employing guard rings and a substrate bias generator to achieve latchup trigger currents in excess of 200 mA. Electrostatic discharge (ESD) protection circuitry to withstand voltages greater than 2001 V and voltage supply tolerances of 10% are standard features of the 7C9116/7 devices that also contribute to its reliability and performance.

System Timing

In microcoded systems there are two loops that determine system performance. These two loops are the data and control loops. The control loop, as shown in *Figure 3*, is essentially the instruction stream for the 7C9116/7.

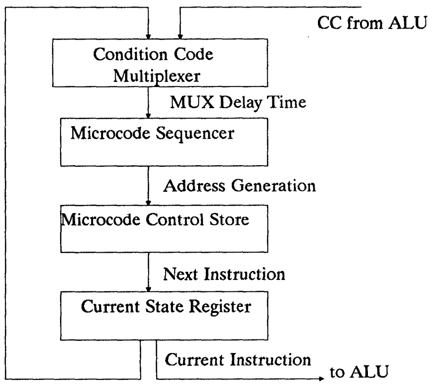


Figure 3. Microcoded System Control Loop

The current instruction combined with other status information is used to generate a new address and instruction for the processor. The data loop, shown in *Figure 4*, moves information from an external source to a

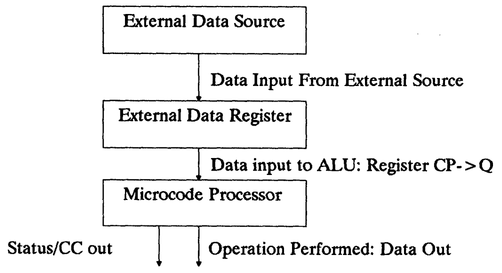
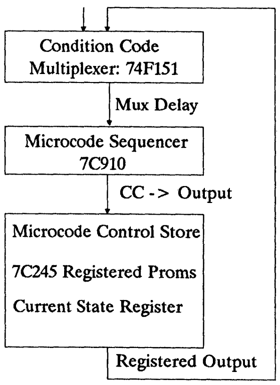

External Data Source
Data Input From External Source
External Data Register
Data input to ALU: Register CP->Q
Microcode Processor
Status/CC out
Operation Performed: Data Out

Figure 4. Microcoded System Data Loop

register where it is stored and operated on by the 7C9116/7 to produce a result and status information for use by the external element. It should be apparent that this is a Harvard-style architecture as instructions and data are in separate domains. Hence, to achieve optimal performance, each of these loops should be as short as possible and equal in length.

An example of control loop timing for a typical 7C9116/7 system is shown in *Figure 5*. Four 7C245A,

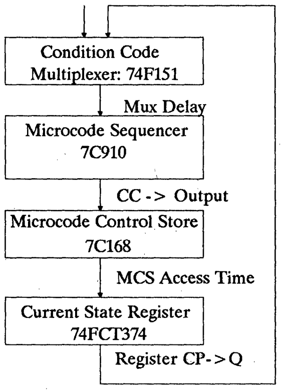


Mux Delay	9ns
7C910 CC -> Output	22ns
7C245A Setup Time	12ns
7C245A CP -> Q	18ns
Total	61ns

Figure 5. Control Loop Timing for Embedded Applications

registered 2K x 8 PROMs are used to implement the control store and current state register in a single package. The 7C910 12-bit microsequencer is used to allow for 4K words of addressing, i.e., instruction memory. In this example a 74F151 is used to multiplex status and condition code information into the sequencer to complete the control loop. The components that make up this particular system would be appropriate for embedded applications where the microcode control store is fixed.

Improved system performance and flexibility can be achieved by using Cypress static RAMs instead of PROMs, thus forming a writeable control store (WCS). As diagrammed in *Figure 6*, four 7C168, 4K x 4 static


Condition Code Multiplexer: 74F151
Mux Delay
Microcode Sequencer 7C910
CC -> Output
Microcode Control Store 7C168
MCS Access Time
Current State Register 74FCT374
Register CP->Q

Mux Delay	9ns
7C910 CC -> Output	22ns
7C168 Access time	20ns
74FCT374 CP -> Q	6.5ns
Total	57.5ns

Figure 6. 7C9116/7 Reprogrammable Control Loop Timing

RAMS can replace the ROMmed microcode control store. However, an external 74FCT374A register must be added to make up for the on-chip register of the 7C245A PROM. Thus, there is a board space penalty for slightly improved performance and increased flexibility. Flexibility is defined as the microcode's abilility to be downloaded or reprogrammed at run time to allow different applications or algorithms to be loaded into the machine as needed by the user or system designer.

The data loop timing for both the embedded and reprogrammable microcoded applications is shown in *Figure 7*. Here, the 7C9116/7 and its fast operation benefit the systems designer in two ways. First, as the data path is significantly faster than the control path, results are available early for the external data units,

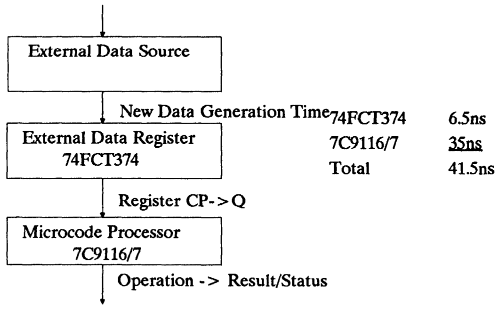


Figure 7. Microcoded System Data Loop Timing

thereby allowing more time for external operations. Secondly, as faster memory technologies become available, systems can be designed to operate at rates up to 25 MIPS.

Applications - Old and New

The applications for fast CMOS 16-bit microprogrammed ALUs can be divided into two categories. The first is similar to their traditional use as a central processing unit for general purpose computing. A designer may choose to use a microprogrammed machine simply because instruction set compatibility with previous machines may be a design requirement. Here, the 7C9116/7's speed and low power serve as powerful upgrades to existing hardware, with the possibility of lower cost from reduced power supply needs.

The more exciting applications for 16-bit microprogrammed ALUs are in loosely coupled coprocessor or embedded controllers. Here, the 7C9116/7's special bit, rotate, and CRC capabilities deliver significant performance advantages over, "off-the-shelf," microprocessors. Graphics and imaging coprocessors benefit from single clock bit manipulation and rotation. The forward and reverse CRC instructions are very helpful in communications and disk controller applications in terms of speed and code density. Graphics, communications, and disk controllers are just three examples that benefit from an application specific instruction set provided by microprogrammed machines like the 7C9116/7.

There remain a myriad of custom control and embedded applications in military, industrial and commercial systems which exploit the performance and flexibility of the 7C9116/7 CMOS 16-bit microprogrammed arithmetic and logic units.

NOTES:

CYPRESS SEMICONDUCTOR

Understanding FIFOs

Introduction

FIFO is an acronym for First-In-First-Out.

In digital electronics, a FIFO is a buffer memory that is organized such that the first data entered into the memory is also the first data removed from the memory.

History of FIFOs

Software FIFOs

Software FIFOs have been (and are being) used extensively in computer programs where tasks are placed in queues waiting for execution. In the programmers' language the program (process) that puts data into the memory is a "producer" and the program that takes data out is a "consumer". Obviously the producer and the consumer cannot access the memory simultaneously. It is the responsibility of the programmer to insure that contention does not occur. Data transfer via a shared memory is a standard programming technique but it is not feasible to have the processor in the data path for data rates greater than 5 Megabytes per second (MB/s). For higher data rates DMA, FIFO, or some combination of the two techniques are used to transfer information.

Hardware FIFOs

In the design of systems, once procedures are standardized and verified in software, the software can be replaced with hardware. The benefits of doing this are improved performance, reduced software, ease of design and usually reduced costs.

Register Array

The first hardware FIFOs were of the "register array" architecture and included the serializer/deserializer (SERDES) within the IC. As they evolved, and due to the ubiquitous microprocessor, the parallel input and parallel output configuration became the standard. For applications that required SERDES users added external shift registers.

The method of transferring data from one register to another is called a "bucket brigade". The transfer is controlled by a "valid data" bit (one per word) that designates which words have been written into but not yet read from and combinatorial control logic. The time for this logic to propagate a word of data from the input to the output of an initially empty FIFO is called "fallthrough time".

Dual Port Ram

The "second generation" of FIFOs are of the "dual port RAM" architecture. In order to achieve truly independent, asynchronous operation of inputs and outputs, the capability to read and write simultaneously must be designed into the basic memory cell.

The fallthrough time present in the register array organization is eliminated by the RAM architecture. However, the RAM must be (internally) addressed, which requires two pcinters. One points to the location to be written into and the other points to the location to be read from. In addition, a bit is required for every FIFO word to designate which words have been written to but not yet read.

Applications

FIFOs are used as building blocks in applications where equipment that are operating at different data rates must communicate with each other, i.e., where data must be stored temporarily or buffered.

These include:

- Word processing systems
- Terminals
- Communications systems; including Local Area Networks
- EDP, CPU, and peripheral equipment; including disk controllers and streaming tape controllers

The Ideal FIFO

The characteristics of an ideal FIFO are:

INPUTS

- Infinitely variable input frequency (0 to infinity)
- Infinitely variable input handshaking signals

OUTPUTS

- Infinitely variable output frequency
- Infinitely variable output handshaking signals

The Ideal FIFO (Continued)

BOTH

- Inputs and outputs are completely independent and asynchronous to each other, except that over-run or under-run are not possible.

STATUS INDICATORS

- Full/empty
- One-half full, ¼ full, ¼ empty

LATENCY

- The latency should be zero. In other words, the data should be available at the FIFO outputs as soon as it is written. In the empty condition this would be the next cycle.

EXPANSION

- Expandable word length and depth without external logic and without performance degradation.

NO FALLTHROUGH OR BUBBLETHROUGH TIME

Analysis of Present Architectures

Register Array

The first Integrated Circuit FIFOs were an extension of the simplest FIFO of all; a serial shift register.

Input Stage

As illustrated in *Figure 1*, the input stage is a one word by m-bit parallel shift register that is under control of the input handshaking signals SI (Shift In) and IR (Input Ready).

Output Stage

The output stage is also a one word by m-bit parallel shift register that is under control of the output handshaking signals OR (Output Ready) and SO (Shift Out).

Register Array

The middle N-2 X m-bit registers are controlled by signals derived from the preceding control signals.

Valid Data

A flag bit is associated with each word of the FIFO in order to tell whether or not the data stored in that word is valid. The usual convention is to set the bit to a one when the data is written and to clear it when the data is read.

Fallthrough and Bubblethrough

The preceding statements regarding input and output stages are not precisely correct under two special conditions, which occur when the FIFO is empty and full:

EMPTY CONDITION - FALLTHROUGH

In the empty condition the data must enter the input stage and propagate to the output stage. This is called Fallthrough time and it limits the output data rate.

FULL CONDITION - BUBBLETHROUGH

When the FIFO is full and one word is read, all of the remaining words must move down one word (or the empty word must propagate to the input). This is called Bubblethrough time and it limits the input data rate.

As we shall see, Bubblethrough time and Fallthrough time are usually equal because the same logic is used.

Dual Port RAM Architecture

The dual port RAM architecture refers to the basic memory cell used in the RAM. By adding read and write transistors to the conventional two transistor RAM cell, the read and write functions can be made independent of each other. Obviously this increases the size of the RAM cell, but doing this is more than compensated for by simpler control logic and improved performance.

The RAM requires two address pointers; one to address the location where data is to be written and the other to address where data is to be read. Comparators are used to sense the empty and full conditions and control logic is required to prevent over-run and under-run.

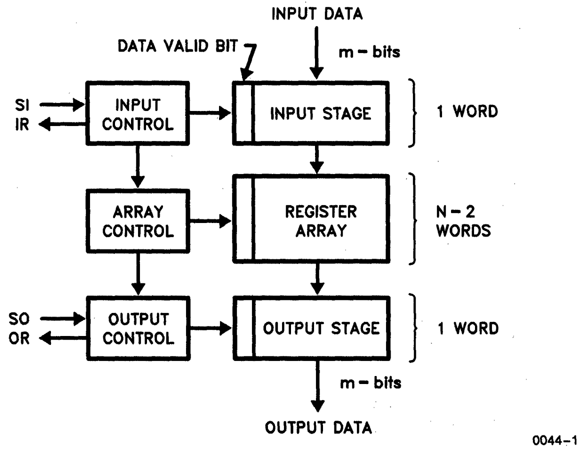


Figure 1. Register Array Architecture

Analysis of FIFOs

The procedure will be to first analyze the FIFO as a "black box" and then to compare the most important characteristics of a class of representative FIFOs with the characteristics of the CY7C401 FIFO.

The class of FIFOs chosen is the industry standard XXX401A and XXX402A that are available from several sources. The 401 is 64 x 4 and the 402 is 64 x 5 with the same performance. Both are of the register array architecture. Both are expandable in depth (number of words), which is called cascadeable, without additional logic as well as expandable in word width (number of bits per word) with additional logic. The operation will first be analyzed in the standalone configuration.

Functional Description

Data Input - Refer to *Figures 2, 3*

After power-on the Master Reset ($\overline{MR}$) input is pulsed LOW to initialize the FIFO. When the IR output goes high it signifies that the FIFO is able to accept data from the producer at the DI inputs. Data is entered into the input stage when the SI input is brought high (if IR is also high). SI going high causes IR to go low, acknowledging receipt of the data, which is now in the input stage.

When SI goes low (in response to IR going low) and if the FIFO is not full, IR will go back high, indicating that more room is available in the FIFO. At the same time SI goes low data is propagated to the next empty location, which may be the second location, but could be any location up to but not including the output stage.

Data Output - Refer to *Figures 4, 5*

Data is read from the DO outputs of the output stage under control of the SO and OR handshaking signals. The high state of OR indicates to the consumer that valid data is available at the outputs. When OR is high, data may be shifted out by bringing the SO line high (request), which causes the OR line to go low (acknowledge). Valid data is maintained on the outputs as long as SO is high. When SO goes low (in response to OR going low) and if the FIFO is not empty, OR will go back high, indicating that there is new valid data at the outputs. If the FIFO is empty OR will remain low and the data on the outputs will not change.

Empty/Full

If the FIFO is empty, OR will not go high within a fall-through time after SO goes low, so this condition may be sensed and used to indicate EMPTY.

Similarly, if the FIFO is full, IR will not go high within a bubblethrough time after SI goes low, so this condition may be sensed and used to indicate FULL.

Standalone Operation

Input Data Setup and Hold

The input data must be stable for an amount of time equal to the setup time (t_{IDS}) before the rising edge of SI and

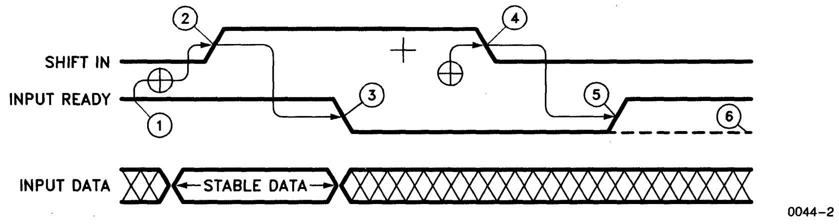


Figure 2. Method of Data Input

Notes:

Shift in pulses applied while Input Ready is LOW will be ignored.

⊕ External "producer" response time.

+ SI pulse could be of fixed positive duration and would then not depend upon response time of producer.

① Input Ready HIGH indicates space is available and a Shift in pulse may be applied.

② Input Data is loaded into the first word.

③ Input Ready goes LOW indicating the first word is full.

④ The Data from the first word is released to propagate to the second word.

⑤ The Data from the first word is transferred to the second word. The first word is now empty as indicated by Input Ready HIGH.

⑥ If the second word is already full then the data remains at the first word. Since the FIFO is now full, Input Ready remains low.

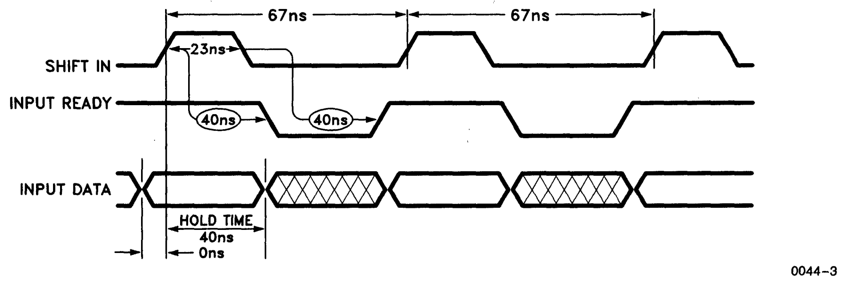


Figure 3. Input Timing for FIFO

Analysis of FIFOs (Continued)

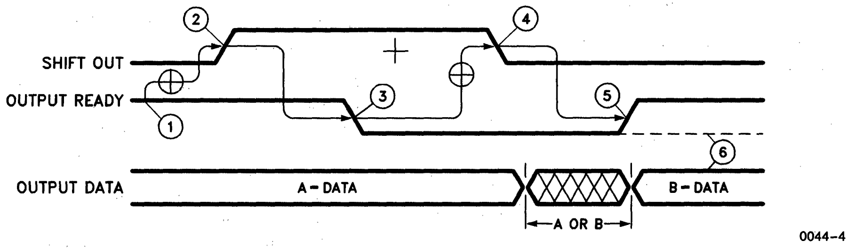


Figure 4. The Method of Shifting Data Out of the FIFO

Notes:

⊕ External "consumer" response time.
+ SO pulse could be of fixed positive duration and would then not depend upon response time of consumer.
① Output Ready high indicates that data is available and a Shift Out pulse may be applied.
② Shift Out goes high causing the next step.
③ Output Ready goes LOW.
④ Contents of word 52 (B-DATA) is released to propagate to word 53.
⑤ Output Ready goes high indicating that new data (B) is now available at the FIFO outputs.
⑥ If the FIFO has only one word loaded (A-DATA) then Output Ready stays LOW and the A-DATA remains unchanged at the outputs.

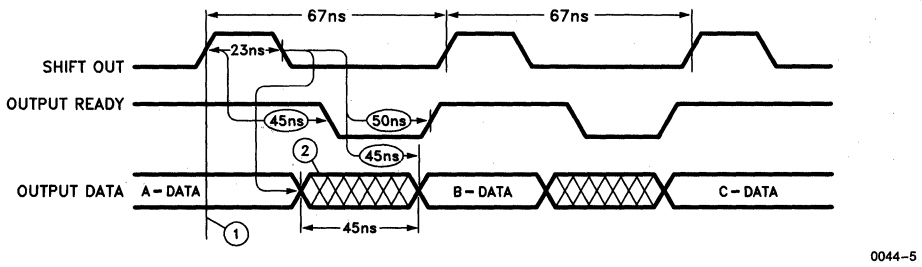


Figure 5. Output Timing for Register Array FIFO

Notes:

① The diagram assumes that, at this time, words 63, 62, 61 are loaded with A, B, C Data respectively.
② Data in the crosshatched region may be A or B Data.

remain stable for an amount of time equal to the hold time (t_{IDH}) after the rising edge of SI.

$t_{IDS} = 0$ ns

$t_{IDH} = 40$ ns

Input Timing

Figure 3 shows the timing relationships between the input data and the handshaking signals when operating at the maximum input data rate of 15 MHz. The Input Ready signal lags (follows) the rising edge of the Shift In signal by 40 ns (max.) for this two edge handshake.

Fallthrough Time

Figure 2 shows the method of entering data into the FIFO. The fallthrough time (*Figure 6*) is measured from the falling edge of the SI signal to the rising edge of the IR signal. For a 15 MHz Register Array FIFO, this time is specified as $t_{PT} = 1.6\ \mu s$ (microseconds).

Register Array Propagation Delay Time

The register array propagation delay time may be approximated by using the delay from the falling edge of the SO signal to the rising edge of the OR signal as being representative of the data propagation delay through the output stage and subtracting this from the fallthrough time.

Reg. Prop. Delay =

Fallthrough time − Output Prop. Delay Time

The delay per stage is then calculated by dividing the register array propagation delay time by the number of stages the data propagates through.

$$\text{Reg. Prop. Delay} = 1.6\ \mu s - 50\ \text{ns} = 1.55\ \mu s$$

$$\text{Delay per stage} = \frac{1.55\ \mu s}{64 - 2} = 25\ \text{ns}$$

Analysis of FIFOs (Continued)

Output Timing

Figure 5 shows the timing relationships between the output data and handshaking signals when operating at the maximum output data rate of 15 MHz. The Output Ready signal lags the Shift Out signal by 45 ns (max.) for this two edge handshake. Data is shifted to the output stage on the falling edge of SO, but does not stabilize until 45 ns later. OR goes low in response to SO going high (45 ns later) and then goes back high 50 ns (max) after the high to low transition of SO.

The reader may assume that the (new) output data is valid 50 – 45 = 5 ns before the rising edge of the OR signal, but this is incorrect. The data sheet specifies these two numbers only as maximums and not also as minimums. Evaluation of these FIFOs has revealed that the data may change several nanoseconds AFTER the rising edge of the OR signal.

The consumer is responsible for delaying the rising edge of the SO signal in order to satisfy his data setup time requirements, which may further reduce the throughput.

Full Condition

The maximum propagation delay from SI going low until IR goes high is 40 ns (*Figure 3*). The bubblethrough time for the full condition is illustrated in *Figure 7*. This time, t_{PT}, is specified as 1.6 μs on the data sheet. The delay per stage is calculated by subtracting 40 ns from 1.6 μs and dividing by the number of stages (64 − 2).

$$\text{Delay per stage} = \frac{\text{Bubblethrough time} - \text{Output Delay time}}{\text{Number of stages}}$$

$$= \frac{1.6\ \mu s - 0.04\ \mu s}{64 - 2}$$

$$= 25.16 \text{ ns}$$

Bubblethrough Time

The bubblethrough timing is illustrated in *Figure 7*. It is seen to be equal to the fallthrough time.

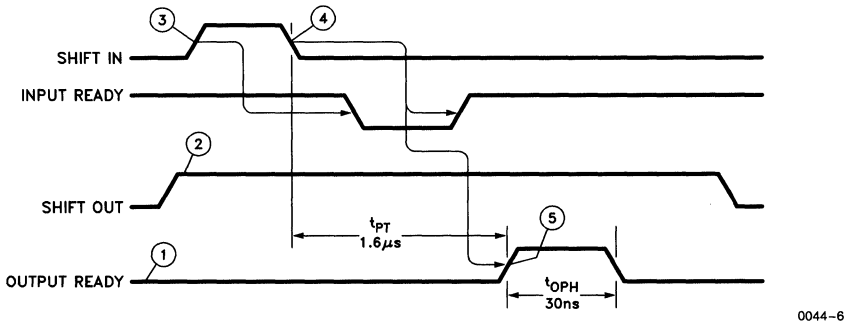


Figure 6. Fallthrough Timing

Notes:
① FIFO initially empty.
② Consumer requests data.
③ Producer enters data.
④ Data enters internal register array.
⑤ Data is available at output.

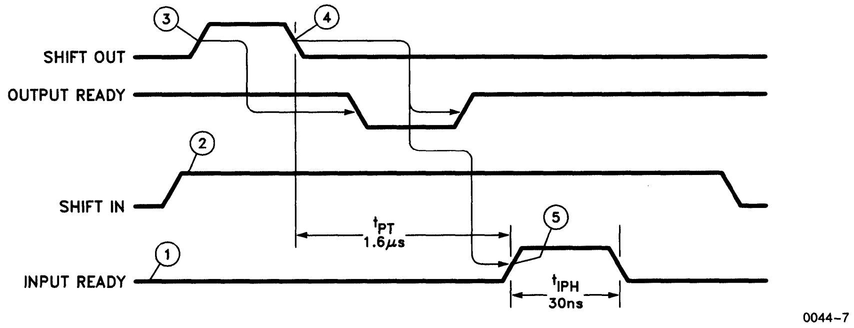


Figure 7. Bubblethrough Timing

Notes:
① FIFO is initially full.
② Shift In held HIGH.
③ Consumer reads data.
④ Empty location begins to propagate to input.
⑤ Empty location reaches input.

Analysis of FIFOs (Continued)

Maximum Throughput Calculations

The maximum throughput of the FIFO is seen to be limited by the fallthrough time when it is empty and the bubblethrough time when it is full.

The "throughput period" corresponding to the "standalone period" (t_A) and the fallthrough time (t_F) is:

$$T_{max.} = t_A + t_F$$

Converting to frequency yields

$$\frac{1}{F_{max.}} = \frac{1}{F_A} + t_F$$

Rearranging and solving for F_{max} yields

$$F_{max} = \frac{1}{\frac{1}{F_A} + t_F} \qquad \text{EQ. 1}$$

The expressions for the throughput frequencies for the FIFO under the full and empty conditions are then;

EMPTY FIFO

$$F_{in} = F_{in\,(max.)}$$

$$F_{out} = \frac{1}{\frac{1}{F_A} + t_F}$$

FULL FIFO

$$F_{out} = F_{out\,(max.)}$$

$$F_{in} = \frac{1}{\frac{1}{F_A} + t_F}$$

The maximum throughput that can be handled by a "nearly empty" or a "nearly full" FIFO operating in the standalone mode is then:

$$F_{(max.)} = \frac{1}{\frac{1}{F_A} + t_F}$$

$$F_{(max.)} = \frac{1}{\frac{1}{15\text{ MHz}} + 1.6\,\mu\text{s}} = \frac{1}{1.667\,\mu\text{s}}$$

$$F_{(max.)} \doteq 599.88\text{ kHz}$$

Note that this is considerably less than the 15 MHz specified on the data sheet.

FULLNESS SENSITIVITY (STANDALONE)

The number of words written into the FIFO corresponding to the fallthrough time if the input data rate is at the maximum (15 MHz) is:

$$\frac{F_{in}}{F\text{ fallthrough}} = \frac{15\text{ MHz}}{\frac{1}{1.6\,\mu\text{s}}} = 24\text{ words.} \qquad \text{EQ. 2}$$

Since the bubblethrough time is the same as the fallthrough time (in this case) the same number of words can be output at the maximum data rate from a full FIFO.

What this means is that the FIFO can operate at its maximum data rate (15 MHz) only when it is between 24 words and 64 − 24 = 40 words full. In order to NOT be sensitive to its fullness, the FIFO must be operated at a maximum frequency less than or equal to the frequency corresponding to the fallthrough/bubblethrough time (625 KHz).

Cypress proposes defining a Fullness Sensitivity (FS) figure of merit for FIFOs that is a measurement of the capacity range (or fullness) over which the FIFO can be operated at its maximum input rate AND its maximum output rate. The FS is normalized; one (1) is ideal and $1 > FS > 0$.

$$FS = \frac{N - F_{IA}\,t_F - F_{OA}\,t_B}{N} \qquad \text{EQ. 3}$$

Where: FS = Fullness Sensitivity
N = The number of words in the FIFO
F_{IA} = Standalone maximum input frequency
t_F = Fallthrough time
F_{OA} = Standalone maximum output frequency
t_B = Bubblethrough time

As an example we will calculate FS for a typical register array FIFO.

$$F_{IA} = F_{OA} = 15\text{ MHz}$$

$$t_F = t_B = 1.6\,\mu\text{s}$$

$$N = 64\text{ words}$$

$$FS = \frac{64 - 15\times10^6\times1.6\times10^{-9} - 15\times10^6\times1.6\times10^{-9}}{64}$$

$$FS = \frac{64 - 24 - 24}{64}$$

$$FS = 0.25$$

If the partial products would have had fractional parts we would have rounded them up to the next highest integers.

FIFO Expansion

The interconnection of two 64 word FIFOs to form a 128 x 4 FIFO is shown in *Figure 8*. Observe that the OR output of the first FIFO becomes the SI input of the second FIFO and that the IR of the second becomes the SO input to the first.

What this means is that the bubblethrough/fallthrough times serially add when the FIFOs are cascaded.

The maximum throughput that can be handled by two FIFOs cascaded together is:

$$F_{(max.)} = \frac{1}{\frac{1}{F_A} + 2\,t_F}$$

$$F_{(max.)} = 306\text{ KHz}$$

Where, as before, $F_A = 15\text{ MHz}$, $t_F = 1.6\,\mu\text{s}$.

Analysis of FIFOs (Continued)

In general, when N FIFOs are cascaded together, the maximum throughput of the combination is:

$$F_{(max.)} = \frac{1}{\frac{1}{F_A} + N\,t_F} \qquad \text{EQ. 4}$$

The FS is also affected by the cascading of FIFOs. If N FIFOs are cascaded together the number of words that can be output or input is N times that of the standalone condition.

$$\frac{F_{in}}{F\ \text{fallthrough}} = \frac{F_A}{\frac{1}{N\,t_F}} \qquad \text{EQ. 5}$$

If this number is greater than the actual (physical) FIFO depth it means that the FIFO cannot be operated at its maximum frequency.

To make a wider word, as well as a deeper FIFO, connect the FIFOs as illustrated in *Figure 9.* Composite IR and OR signals must be generated using two external AND gates (e.g., 74LS08) to compensate for variations in the propagation delay of these signals from device to device. The maximum throughput for this configuration is 205 KHz (N = 3 in preceding formula).

Cascadability Considerations

In order to guarantee the ability of multiple FIFOs to reliably cascade with each other using the handshaking method previously described, certain conditions must be met. These are now considered.

SI or OR Signal Compatability

In the cascaded configuration, the OR signal of the Nth FIFO must be specified such that it can be detected when it is applied to the SI input of the N + 1th FIFO. See *Figure 8.* This means that the minimum high time (positive pulse width) of the OR output signal of the input FIFO must be able to be recognized at the SI input of the output FIFO.

IR and SO Signal Compatability

In the cascaded configuration, the IR output of the N + 1th FIFO must be specified such that it can be detected when it is applied to the SO input of the Nth FIFO.

Minimum Delay Between SI and IR

The minimum delay between SI going HIGH and IR going LOW is an unspecified parameter in the industry standard

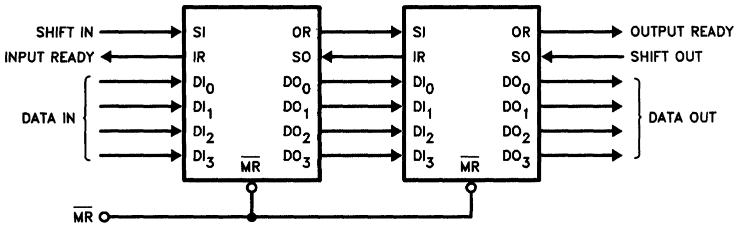


Figure 8. 128 x 4 FIFO

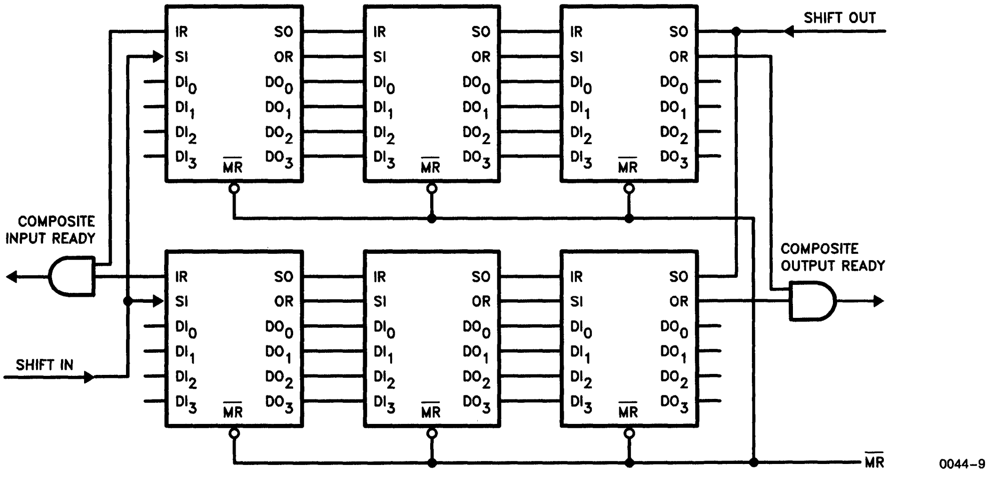


Figure 9. 192 x 8 FIFO

Analysis of FIFOs (Continued)

data sheets. The Cypress FIFO exhibits a 6 to 10 ns minimum delay. Care must be taken when mixing Cypress FIFOs and competitive FIFOs to insure that the parts will cascade with one another. In general, delaying the IR output of the Cypress FIFOs enables competitive parts to cascade with Cypress parts. The Cypress FIFO can always recognize the output of the competitive product.

Minimum Delay Between OR and SO

Another unspecified industry parameter is the delay between OR and SO. The minimum delay for Cypress FIFOs is 6 ns. A 500 pF capacitor added between the OR pin and ground and the IR pin and ground of all Cypress FIFOs will permit cascading with competitive FIFOs. These capacitors delay the signals the appropriate amount of time.

Cascading at the Operating Frequency

In order to operate at a given frequency, F_O, in the cascaded configuration the following relationship must be satisfied;

$$t_{SIH} + t_{IRH} < \frac{1}{F_O}$$

This condition is met by both the MMI and Cypress FIFOs.

Description of the CY7C401

A block diagram of the CY7C401 is shown in *Figure 10.* It is a direct, pin for pin, functional equivalent, improved performance, replacement for the register array FIFOs. The similarities and differences between the 401, 402, 403, and 404 are summarized in the table.

Product	Configuration	t_F	Package	Description
CY7C401	64 x 4	65 ns	16 pin DIP	Industry Standard
CY7C403	64 x 4	65 ns	16 pin DIP	Pin 1 is three-state output enable
CY7C402	64 x 5	65 ns	18 pin DIP	Industry Standard
CY7C404	64 x 5	65 ns	18 pin DIP	Pin 1 is three-state output enable

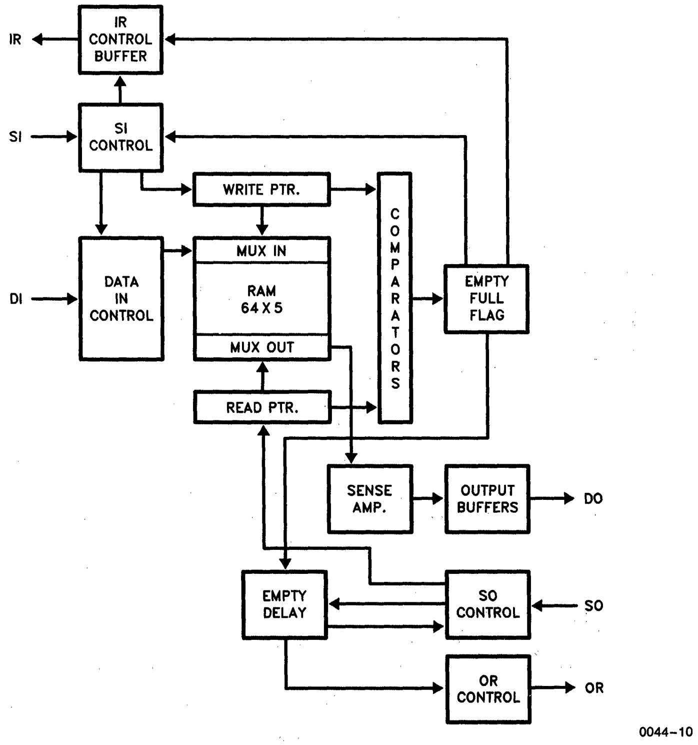


Figure 10. CY7C401 Block Diagram

Description of the CY7C401 (Continued)

Architecture Refer to *Figure 10*

The architecture is that of a dual port RAM, which is accessed by two pointers; a read pointer and a write pointer. The input data and output data do not reside in input or output registers as in the register array architecture. Instead, the pointers address the memory locations of the input and output data. Comparators are used to control the IR and OR lines to prevent overflow and underflow. The key to this architecture is the dual port RAM cell, which is illustrated in *Figure 11*. It is only 1.2 square mils in area. Separating the read and write functions enables the memory cell to be read from and written to simultaneously and independently. This increases the basic cell size, but simplifies the overall architecture and improves the performance.

The bubblethrough time is greatly reduced (65 ns versus 1.6 μs) because it now represents the time required to update the pointers, not the time required for data to propagate through the memory array.

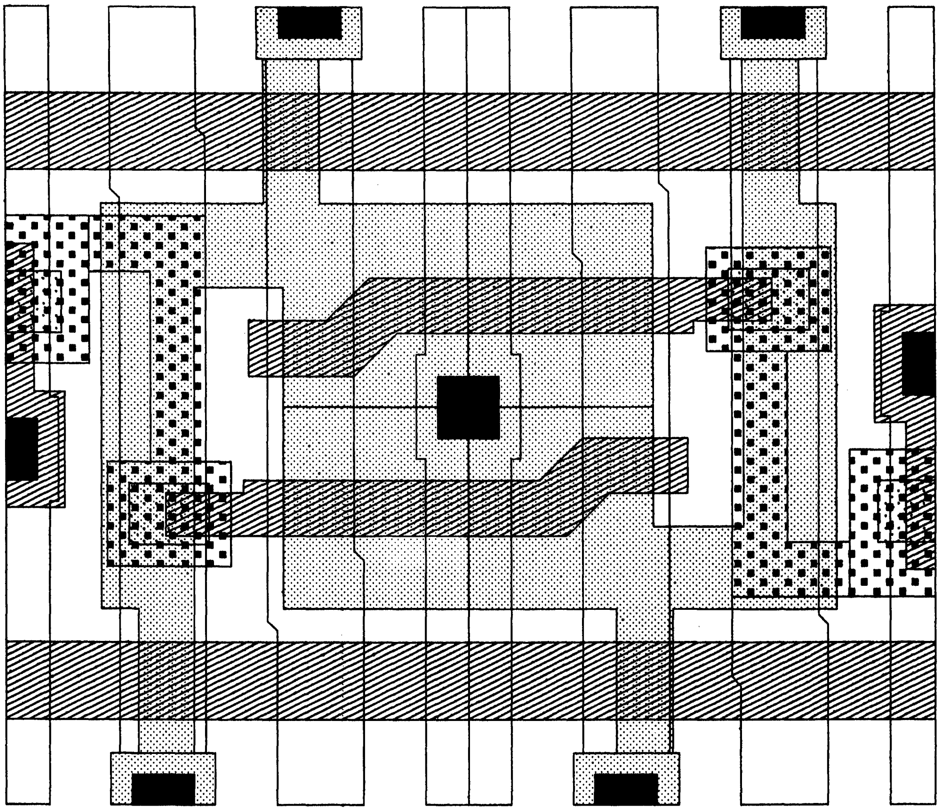
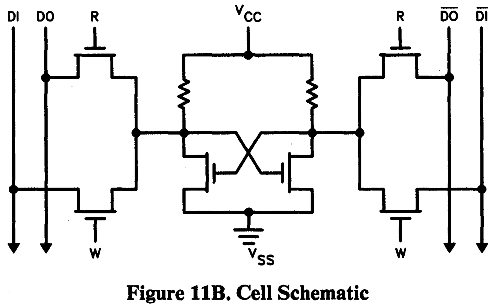

0044-11

Figure 11A. CY7C401 Ram Cell Layout



0044-12

Figure 11B. Cell Schematic

Description of The CY7C401 (Continued)

Functional Description

To the "outside world" the CY7C401 appears functionally equivalent to the register array FIFOs. All of the timing diagrams as well as the expansion diagrams of *Figures 8* and *9* apply.

Input data is sampled with the rising edge of the SI signal if the IR signal is high. The input (write) pointer is incremented on the falling edge of the SI signal.

Data is output with the falling edge of the SO signal if the OR signal is high. The output (read) pointer is incremented on the rising edge of the SO signal.

Output Timing

In the discussion on output timing it was pointed out that (for the register array FIFO) the way the timing of the data out with respect to OR, there is no guarantee that the data will be stable before the rising edge of OR. This time (t_{SOR}) is guaranteed to be a minimum of 5 ns on the CY7C401 data sheet.

Comparison of Register Array FIFOs and the CY7C401

Throughput

Using equation 4 the values in the following table were calculated and are plotted in *Figure 12.*

Fullness Sensitivity

Register Array FIFOs in the Standalone Mode

Equation 2 was used to calculate the number of words that could be input and output corresponding to the maximum frequency of 15 MHz. Subtracting these from the FIFO capacity (64) gives us the capacity range over which the FIFO can operate at its maximum rate. This was calculated to be between 24 and 40 words, or 32 ± 8 words. Equation 3 was used to calculate the FS and it was found to be 0.25.

Using equation 2 we have;

$$\frac{F_{in}}{\text{F fallthrough}} = \frac{F_A}{\frac{1}{t_F}}$$

$$= \frac{15\text{ MHz}}{\frac{1}{67\text{ ns}}} = 0.975\text{ words}$$

The CY7C401 is seen to be much less sensitive to fullness than the register array FIFOs. Its capacity can range from 2 to 63 words, or 32 ± 31 words in the standalone mode.

The Fullness Sensitivities are plotted in *Figure 13.* They are also plotted in a slightly different form in *Figure 14.*

A little thought will convince the reader that Fullness Sensitivity is another way of quantifying the range of the difference between input and output data rates. The closer the FS is to 1 the greater the capacity of the FIFO to handle bursts of data.

Latency

The classic definition of latency is the difference, in elapsed time, between when a resource is requested and when it is granted. In disks, the worst case latency is the time required for one revolution of the disk. The average latency is then the time required for one-half a revolution. The assumptions are one head per track and no contention for the head.

Worst Case Latency - refer to *Figures 6* and *7*

The worst case latency for the consumer occurs when the FIFO is empty and for the producer when it is full. It is;

Where: $t_{in} + t_{out} + t_F$

t_{in} = period of the input frequency

t_{out} = period of the output frequency

t_F = Fallthrough time

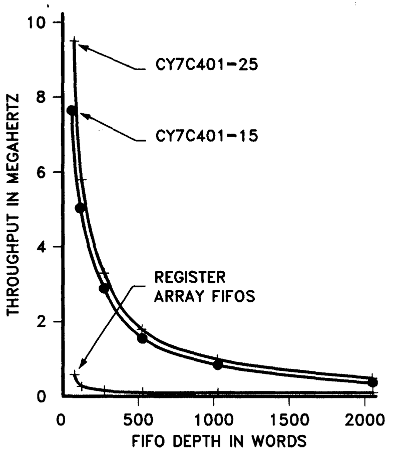
Average Latency

If the FIFO is operated such that it is not sensitive to its fullness $t_F = 0$. In addition, if $t_{in} = t_{out}$ the average latency is one cycle. Otherwise, it is;

$$\frac{t_{in} + t_{out}}{2}$$

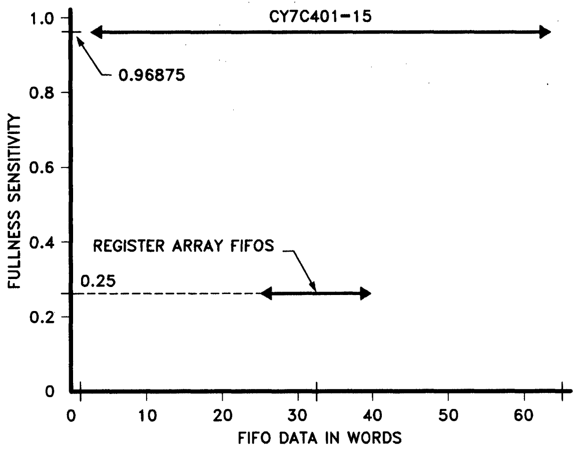
			Throughput		
	N	D	C67401A	CY7C401-5	CY7C401-25
F_A	—	—	15 MHz	15 MHz	25 MHz
t_F	—	—	1.6 μs	65 ns	65 ns
	1	64	600 KHz	7.57 MHz	9.52 MHz
	2	128	306 KHz	5.01 MHz	5.8 MHz
	4	256	155 KHz	3 MHz	3.3 MHz
	8	512	77.7 KHz	1.7 MHz	1.78 MHz
	16	1024	38.9 KHz	903 KHz	925.9 KHz
	32	2048	19.5 KHz	465.7 KHz	471 KHz

Comparison of Register Array FIFO's and the CY7C401 (Continued)



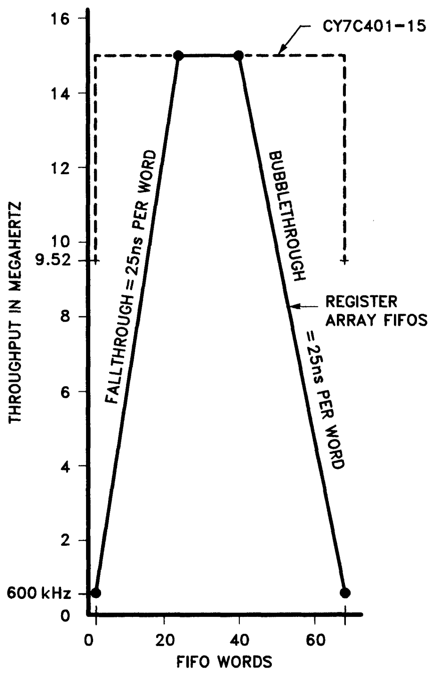
0044-13

Figure 12. Maximum FIFO Throughput vs. Depth



0044-14

Figure 13. Fullness Sensitivity in the Standalone Mode



0044-15

Figure 14. Standalone Throughput

Summary and Conclusions

In most systems where FIFOs are used they are neither full nor empty, except at the beginning or end of an operation. After analyzing the preceding two FIFOs the reader can understand why. Serious performance degradation occurs under these conditions, especially if the FIFO uses the register array architecture. To compensate for this, manufacturers have added one-half empty/full indicators (etc.), which has helped by alerting the system controller before the performance suffers.

A better solution to the performance problem is to use a FIFO that has the dual port RAM architecture, which has been shown to result in a superior performance FIFO.

CYPRESS SEMICONDUCTOR

Interfacing to the FIFO Application Brief

Introduction

This application brief is intended to be a guide to the FIFO user and to make him aware of certain conditions that should be considered when interfacing to the FIFO. The two areas of concern are (1) voltage sensitivity on the SI and SO inputs and, (2) metastability when the SI or the SO signals are derived from independent clocks. These two issues are independent of each other. All comments apply to the following Cypress CMOS FIFOs: CY7C401/402/403/404, CY7C3341, CY7C408/409.

High Gain Inputs

The minimum positive SI and SO pulse widths are specified on the FIFO data sheet as 11 ns (25 MHz SI/SO) and 20 ns (other speed grades). At room temperature and nominal (5V) V_{CC} the FIFO will operate reliably with SI/SO pulses as short as 5 ns. The reason these FIFOs respond to such short pulses is that the Cypress high performance CMOS process yields circuits that have very high gains and, consequently, require very little energy to change state.

Termination networks are recommended on the SI and SO lines (traces) on Printed Circuit Boards (PCBs) when the lines exceed seven inches in length (from source to load). The termination matches the load impedance to the characteristic impedance of the PCB trace, which is typically 50Ω or less for microstrip or stripline construction on G-10 glass epoxy material. For minimum voltage reflections a slightly overdamped termination is preferred. Cypress recommends a series capacitor of 47 pF and resistor of 47Ω be connected from the input pin (SI/SO) to ground as shown in *Figure 1*. This termination network acts as a low pass filter for short, high frequency pulses and dissipates no DC power. If more than one FIFO is connected in parallel to make a wider word only one termination network is required. It should be located at the input that is electrically the farthest away from the source.

Please refer to the low pass filter analysis in the "Systems Design Considerations When Using Cypress CMOS Circuits" application brief (in this databook) for the method of determining the values of R and C for the termination network.

Synchronous And Asynchronous Operation

When the SI and SO signals are derived from a common frequency source (or clock) the FIFO is, by definition, operating in the synchronous mode. There is a precise, known relationship between the SI and SO signals.

Conversely, when the SI and SO signals are derived from two independent frequency sources, the FIFO is operating in an asynchronous mode.

In the synchronous mode the designer can assure that the OR signal not occur within the setup and hold time window that normally "surrounds" the output system clock edge (or sampling signal). The same reasoning applies to the occurance of the IR signal with respect to the input system clock.

In the asynchronous mode, the designer cannot assure a known relationship between the OR signal and the output system clock either with respect to frequency or with respect to phase. It is the responsibility of the designer to insure that, even though the output system clock edge may occur at the same time that the OR signal occurs, the FIFO still receives a SO clock that is wide enough to be reliably recognized as such by the FIFO. The same reasoning applies to the SI signal that is generated in response to the IR signal under control of the input system clock.

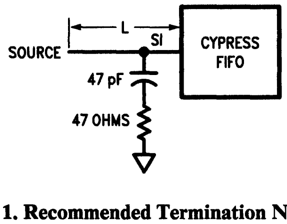


Figure 1. Recommended Termination Network

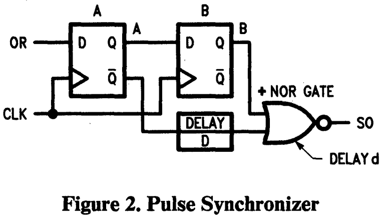


Figure 2. Pulse Synchronizer

November 1986

Pulse Synchronizer

The circuit of *Figure 2* is recommended to generate the SO pulse as a function of OR under control of the output system clock. An identical circuit should be used to generate the SI pulse as a function of IR under control of the input system clock. If it is required to perform control functions on the OR or the IR signals, it should be done before they are clocked by the first D flip-flop.

State Diagram

The two stage shift register is analyzed as a state machine in *Figure 3*. Other, more complex state machines can be designed, but the idea is the same; reliably generate a single pulse of a known minimum width for every OR or IR LOW to HIGH transition.

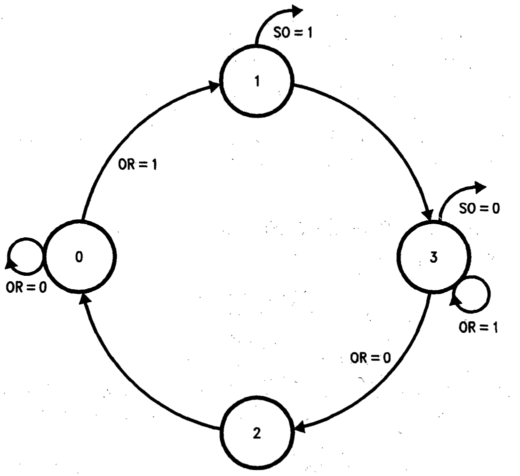


Figure 3. Pulse Synchronizer State Diagram

Transition Table

A	B	STATE	DESCRIPTION
0	0	0	IDLE AT STATE 0
1	0	1	GENERATE SO = 1
1	1	3	GENERATE SO = 0
0	1	2	TRANSITION STATE

0097-4

Design Considerations

The frequency of the clock to the pulse synchronizer should be at least twice that of the maximum rate data is shifted into or out of the FIFO.

For example, if it is required to shift data into the FIFO at a 10 MHz (SI) rate, the clock to the input pulse synchronizer should be 20 MHz. If it is required to shift data out of the FIFO at a 15 MHz (SO) rate the clock to the output pulse synchronizer should be 30 MHz.

Minimum SI/SO Pulse Width

The minimum pulse width of the SO signal of *Figure 2* under normal operating conditions will be one cycle of the output clock (CLK). However, when the OR or the IR signal changes within the "unallowed window" around the clock edge, defined by the flip-flop setup time and hold time, the flip-flop may go into a metastable state. i.e., its outputs may be between the logic ONE and the logic ZERO voltage levels. The amount of time the flip-flop will stay in the metastable region will be approximately 4 X, where X = clock to output propagation delay time.

The minimum pulse width of the SO signal is determined by the delay, d, through the NOR gate, plus any delay the designer may add (D, shown as a box) in the path from the /Q output of the A flip-flop to the input of the NOR gate. The NOR gate acts as a low pass filter and will not pass a pulse if its width is less than d. Adding an external delay, D, increases the minimum pulse width to d + D. The maximum frequency that the circuit can operate at, assuming equal gate turn-on and turn-off times, is then

$$f\,(\text{max.}) = \frac{1}{2\,(d + D)}.$$

The total delay should be chosen such that the minimum pulse width is sufficient to reliably be detected by the FIFO. The preceding comments apply to lumped delays, not to analog or distributed delay lines.

Implementation Of The Delay

If only the NOR gate provides the delay, the following table lists typical and maximum propagation delays under nominal V_{CC} and loading (20 pF) conditions.

Table 1. Propagation Delay in ns

Family	Typical	Maximum
LS	10	15
ALS	5	11
HCMOS	8	23
FACT	5	9.5

A 74LS02 NOR gate will result in a minimum pulse width of 10 ns, which will reliably operate a 25 MHz CY7C403 or a CY7C404 FIFO.

If it is required to operate a 10 MHz CY7C401/402, the Q output of the A flip-flop may be inverted through a 74LS04 and applied to the lower input of the NOR gate. The minimum pulse width is then 10 + 10 = 20 ns.

A delay line or a RC network could also be used to delay the signal to the lower input of the NOR gate.

The circuit of *Figure 2* can also be used to synchronize the SI and SO inputs of the CY7C3341.

The rising edge of the SO signal should be used to sample the FIFO data.

CYPRESS SEMICONDUCTOR

Understanding Dual-Port RAMs

Introduction

A dual-port RAM is a Random Access Memory that can be accessed simultaneously by two independent entities. In digital integrated circuits, this implies a dual port memory cell that can be accessed at the same time using two independent sets of address, data, and control lines.

This applications brief examines the evolution of multiport memories and presents, illustrates, and explains the operation of, and benefits to the user of, Cypress's dual-port RAMs.

A Brief History of Multi-Port Memories

The first multi-port memories were probably used in the CPU (Central Processing Unit) of the first computers. Many two-operand instructions are efficiently implemented using dual-port registers for the operands and the result.

Consider, for example, Equation 1, which describes a typical two-operand operation in the ALU (Arithmetic Logic Unit) of a CPU.

$$ (C) = (A)\,[\,\text{OPERATOR}\,]\,(B) \qquad \text{Equation 1} $$

A and B could be either the operands (i.e., the data) or the addresses of the operands, in which case the data could be either in memory or in registers. In any case, *Equation 1* describes two pieces of data, A and B, being operated upon by the OPERATOR, which could be arithmetic or logical, and the results being designated as C. C could also be the data, a register, or a memory location.

The Combinatorial ALU

The 74181 was the first integrated circuit ALU. The four-bit operands, A and B, are operated upon according to a four-bit command, and the result, C, is output. A carry-in input, a carry-out output, and A = B outputs are also provided. A mode control pin selects either logical or arithmetic operations. The 74181 is combinatorial; no storage is provided.

Early computers used the contents of a memory location as one operand and an accumulator in the CPU as the second operand. The results were usually stored in the accumulator.

Bringing the Registers On-Chip

The 67901 was the first four-bit slice that brought sixteen four-bit registers onto the chip. The MMI 67901 was second sourced by AMD and became the 2901. At one point in time there were five sources for this industry standard bipolar ALU. The Cypress CMOS CY7C901 is the highest performance, TTL compatible, four-bit slice that is form, fit, and functionally equivalent to the original **901.

The sixteen-word deep, four-bit wide register array is functionally equivalent to a 16 x 4 dual-port memory. Four A address lines and four B address lines select the contents of two of the sixteen registers, whose outputs are applied to transparent latches. The outputs of the latches are then applied to 3:1 multiplexers, whose outputs drive the ALU inputs. The outputs of the ALU may be sent off chip, entered into a temporary register (Q), or written back into the register file, thus replacing one of the operands. This architecture is shown in the block diagram of the CY7C901 in the Cypress databook and is not reproduced here. Instead, a simplified block diagram of the **901 dual-port memory is presented in *Figure 1.*

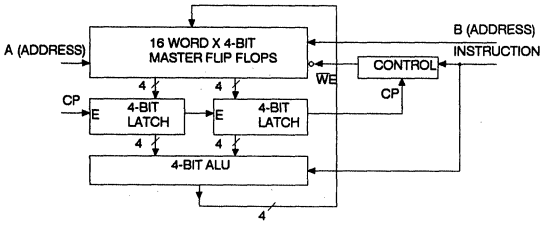


Figure 1. **901 Dual-Port Memory (Simplified)

Functional Operation of the **901 Dual-Port Memory

The A and B addresses select the contents of two registers, whose outputs are applied to two 4-bit latches. When the clock (CP) is HIGH the outputs of the latches follow their data inputs (i.e., are transparent). When the clock is LOW the outputs of the ALU are written ($\overline{\text{WE}}$) into the register array at the location specified by the A or B addresses, depending upon the instruction being executed. The LOW level of the clock causes the data in the latches not to change so that the ALU outputs will be stable when they are written back into the register array.

Observations

It is seen that the **901 dual-port memory does not perform the function described by *Equation 1*, which is a three-port operation. The C operand is equal to either the A operand or the B operand, depending upon the instruction being executed. In fact, the A and B addresses could be the same. An old programming trick is to "exclusive OR" the contents of a register with itself, which clears it.

Also, the dual-port memory of the CY7C901 does not use a dual-port memory cell. It is not required because the operations performed by the **901 do not require simultaneously writing independently to two separate memory locations.

Dual-Port Memory Using a Single Port RAM

Before the dual-port memory cell existed, a standard solution to creating a dual-port RAM using a standard single-port RAM, was to add a multiplexer between the RAM and the two entities which shared the RAM.

A block diagram of such an arrangement is illustrated in *Figure 2.* The RAM is shared between two processors, MP1 and MP2. If each processor has access to it one-half of the time, the resource is shared equally, and it is said to be allocated according to a "fairness" doctrine.

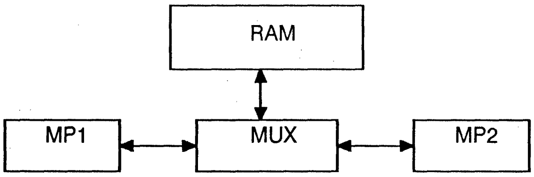


Figure 2. Dual-Port Ram Memory Using a Single-Port Ram

This time division multiplexing assures that there is no contention for the RAM. However, performance suffers if the access time of the RAM is not at least equal to one-half (or less) of the clock period of the microprocessors, assuming that they are clocked from the same source.

For example, if both processors are clocked from the same 25 MHz frequency source (period of 40 ns), and are closely coupled (i.e., there is one and only one operating system in memory), the maximum access time of the dual port will have to be 20 ns or less. The fastest speed dual port available has a 25 ns access time. Therefore, each processor will suffer a worst case 20% performance degradation.

Dual-Port RAM Applications

The first applications for dual-port memories were for CPU register files. They may also be used for cache (data or instruction) memories. However, the largest usage of dual-port RAMs is in communications.

For the purpose of this discussion, communications is further divided into the exchange of data between processors, processes, and systems.

Communications Between Systems

Data is usually exchanged serially in order to reduce the cost of the circuits required. In all real world systems, noise is present and data will be corrupted. Studies have shown that noise on communications channels tends to occur in bursts. Several general techniques are used to improve the probability of correctly receiving data over the channel.

The most basic technique is to divide the data into blocks in order to reduce the probability of a noise "hit". As the block size increases, the probability of getting hit also increases, so the block size should not be too long. However, a small block size leads to inefficient use of the channel.

A second technique is to add additional characters to each block that either (1) guarantee the integrity of the data, such as a CRC (Cyclic Redundancy Character), or (2) detect and correct any errors, such as an ECC (Error Checking and Correcting) field. However, adding characters increases the block size and the probability of a hit, which defeats the purpose of adding the CRC or ECC field.

A third technique is to require a protocol such that acknowledgment of the reciept of a packet (a number of blocks) is required before the next packet is transmitted. This decreases the channel efficiency because it must be "turned around", but only the blocks of data that were corrupted need to be retransmitted.

In many applications all three of the preceding techniques are used. In addition, some channel controllers adaptively change the baud rate, depending upon the error rate.

Virtual Dual-Port RAM

Physical dual-port RAMs are not required for communication between systems. Instead, a conventional RAM memory is partitioned into virtual data storage areas (buffers); usually at least two packets of data are stored. These buffers are shared between the intelligence (usually a microprocessor) that assembles the packets and stores them and the communications controller (which may also be a microprocessor) that reads the data from memory, converts the data from parallel to serial, encodes it, converts it to analog, and sends it out over the communications channel (on the transmit side). If there is only one processor in the system, the data buffers are not shared, and there is no need for either a virtual or a physical dual-port RAM.

Associated with each data buffer is control information that tells the communications controller (1) how many words are in the buffer, and (2) the starting address of the data in the buffer. The control information is stored in one or more RAM memory locations whose addresses have been previously agreed upon by the two processors.

A second level of control is required in this simple example. It is a mechanism or procedure to follow which insures that the two microprocessors do not "get in each other's way". In other words, it is a "procedure control mechanism". Another way of analyzing the procedure that must be followed in order to eliminate contention introduces the concept of "the ownership of data".

In this example the processor that is assembling and storing the messages (let's call it MP A) can be said to "own" the data while it is performing its task. Likewise, the communications processor (let's call it MP B) owns the data while it is performing its task. The procedure reduces to the transfer of ownership of the data between MP A and MP B. In large systems, where many processors perform many different operations, the processing of the information is called a job, or a procedure. The procedure is divided into many tasks, which may be performed by different processors and which may either (1) be scheduled and assigned by a processor dedicated to that task (called an autocratic system), or (2) be performed by any available processor (called an egalitarian [meaning equal processors] system). In either case, the two (or more) processors must have access to a shared memory location that is used for message passing.

Synchronization of sequential processes is the cornerstone of concurrent programming, be it within a multi-tasking single processor system, a distributed network of processors, or a tightly-coupled multi-processor system.

Message Passing Using a Shared Memory and Lockvariables

In the two-processor example under consideration, synchronization can be achieved by the use of a "lock-word" or "lockvariable". The lockvariable can apply either to data (this example) or to executable instructions.

The lockvariable is a location in shared memory that is operated upon using two synchronization primitives; LOCK (v) and UNLOCK(v). They are simple binary switch operations. If a processor wishes to "lock" or own a critical section (code or data) it does so by testing the lockvariable and indivisibly setting it if it was zero. If it was not zero, then the operation is repeated until it is zero. A processor unlocking the critical section simply sets the lockvariable to zero and continues.

Most modern processors have indivisible read/modify/write instructions, also called "test and set" (TAS) instructions. However, Dijkstra's original paper, "Solution of a Problem in Concurrent Programming Control" [1], shows that lockvariables can be implemented without using a read/modify/write instruction. He also developed the semaphore [2], which is a technique for managing a queue of tasks waiting for a resource. Lockvariables "surround" or "bracket" semaphores and thus provide entry and exit control on a mutual exclusion basis.

For the purpose of this example we will assume that the processors have a TAS instruction. A typical TAS instruction operates as follows:

Typical TAS Instruction

Read, test, and set to X. The addressed memory location is read, and if its contents are zero, the value X is written into that location. If the contents are not zero, the contents are returned to the processor and the value in the memory location is not disturbed.

The usual convention is that a value of zero in the lock-variable means that the resource associated with it is available. A non-zero value means that another processor temporarily owns it and that it is not available. After performing the task associated with the lockvariable, the processor sets its value to zero. The system is initialized with all lockvariables set to zero.

In this example, MP A performs a TAS operation on the lockvariable and, finding it zero, sets it to a one. This tells MP B that the message is in the process of being assembled in the memory buffer area and is "not ready" to be transmitted. MP A then assembles the message. After the message is assembled, MP A clears the lockvariable and sends a message to MP B telling it (1) that the message is ready to be transmitted, and (2) where the data is stored and how many bytes are to be sent. MP B reads the message from MP A, performs a TAS operation on the lockvariable and, finding it zero, sets it to a two. This tells MP A that the message is in the process of being transmitted. MP B then transmits the message and clears the lockvariable. MP B then sends MP A a message that the transmission task has been completed. After receiving the message from MP B, MP A performs a TAS operation on the lockvariable and, finding it zero, concludes that the message has been successfully transmitted.

Note that this procedure does not require the use of a dual port RAM. It does require each processor to perform a TAS instruction, clear the lockvariable, and send a message to the other processor. Sending a message implies writing to a location in shared memory. In order to know that a message is waiting, the processor receiving the message must either poll (periodically read) the memory location (mailbox) or, the act of writing to the mailbox must generate an interrupt to the receiving processor. The interrupt driven alternative is usually preferred because the receiving processor does not have to waste time in a polling sequence.

Dual-Port Memory Using A Dual-Port RAM Cell... Recent history

The first dual-port RAM integrated circuits to use a dual-port RAM cell were the Synertek SY2130 and SY2131, which were introduced in 1983. These products were organized as 1024 words of 8-bits and use n-channel double-polysilicon technology to achieve 100 ns access times. The SY2130 had an "automatic power down" feature that was controlled by the chip enables, and the SY2131 did not. The smaller (512 X 8) SY2132 and SY2133 were similar (but unsuccessful).

These original dual-port RAMs included two mailboxes for message passing that, when written to from one port (side), generated an interrupt to the opposite port. Also, on-chip arbitration logic generated a "busy" signal to the loser when both left and right ports addressed the same memory location. If the loser was attempting to write, the write was supressed. Most of the the dual-port RAMs on the market today are functionally equivalent to the original Synertek products. The "new features" that have been added to several dual-port RAM products by Motorola and Integrated Device Technology (IDT) include dedicated "semaphore registers". However, not only are these semaphores unnecessary, the products that use them do not have second sources.

The SY2130 was second sourced by IDT in 1984 and Advanced Micro Devices (AMD) in 1985. IDT also doubled the density to 2K X 8 and called the new part the IDT7132. Due to pin limitations (48 pins), the interrupt functions were deleted.

The AMD part (Am2130, 1024 X 8) had at least three logic errors. Busy going active failed to reset the interrupt when both ports addressed the same mailbox location. Also, busy going inactive failed to retrigger the ATD (Address Transition Detection) circuitry (at all locations). And finally, when contention occurred and both ports were attempting to write, the losing port was not prevented from writing. These conditions are not explicitly stated on the data sheets, but a little thought will convince the reader that these things must be done in order to make logical sense. More about this later.

In 1985 IDT added the "slave" companion parts to their dual-port family. The IDT7140 (1024 X 8) is the slave to the IDT7130 and the IDT7142 (2K X 8) is the slave to the IDT7132. The slave device is used in word width expansion. Busy is an input to the slave (from the master) and there is no arbitration logic in the slave. One master may drive many slaves. This avoids the classic "deadly embrace" problem described in the following paragraphs.

The Deadly Embrace as a Result of Busy Arbitration

If two masters are connected in parallel to make a wider word and if the left and right port addresses match, and if the left and right port chip enables (to both) then become active at approximately the same time, it is possible to have one port of one master lose and the opposite port of the other master also lose. In other words, there is a small "time window" or "aperature of uncertainty", if within which an address match occurs and both ports are enabled, the dual-port RAM cannot determine which port will win or lose.

If the corresponding left and right port busy pins are connected together (respectively), under the preceding conditions, both ports of both masters will be active (LOW). This will happen because the busy outputs are open drain, and the loser will pull the node LOW.

At this point, as far as the external world is concerned, both ports are busy and the system will remain "locked up" indefinitely, with each port waiting to be released by the other. This condition is the simplest example of the deadly embrace. Each "arbiter section" of each master thinks that it has lost the arbitration, and is waiting to be released by the other.

The Classic Definition of the Deadly Embrace

The deadly embrace is a condition that occurs in a system where a processor requires one or more resources in order to perform a task, and one or more of the required resources is temporarily owned by another processor that requires one or more of the same resources in order to perform its task.

For example, if processor A owns resource X and processor B owns resource Y, and both resources are required to accomplish the task, we have a stalemate condition where each processor is waiting for the other to relinquish the required resource. This is the most simple example. The concept can be extended to n processors and m resources.

There is, of course, a solution to this situation. It depends upon whether the system is autocratic or eglitarian, the priorities of the tasks, etc., and is beyond the scope of this discussion.

The Cypress Dual-Port RAM Family

The members of the Cypress dual-port RAM family are tabulated in *Table 1.* The package designator D26 stands for 600 mil ceramic, dual in-line package (DIP) and P25 stands for 600 mil plastic DIP. The 48-pin ceramic leadless chip carrier (LCC) is designated as L68. The 52-pin packages are designated as L69 for ceramic LCC and J69 for plastic LCC (PLCC).

Note that the interrupt function is not available at the 2048 X 8 level in a 48-pin package. This is due to pin limitations. At the two kilobyte level, two additional address pins are required, one for each port, for the address MSB.

Masters and Slaves

The difference between masters and slaves is that the masters have arbitration logic and the slaves do not. The busy signals are outputs from the master and inputs to the slave. The ramifications of this will be examined later.

Dual-Port RAM Block Diagram

A simplified block diagram of the Cypress dual port RAM is shown in *Figure 3.* At the device interface there are two sets of three groups of signals: address, data, and control. By convention, the two sets of signals are called "left port" and "right port". Each and every signal has either the subscript "L" for Left or the subscript "R" for Right.

Address Pins

The address pins are designated A0 through A9 (1024 X 8) and A0 through A10 (2048 X 8), where A0 is the least significant bit (LSB) and A9 or A10 is the most significant bit (MSB). The address pins are unidirectional inputs to the device; their states specify the memory location to be either read from or written into.

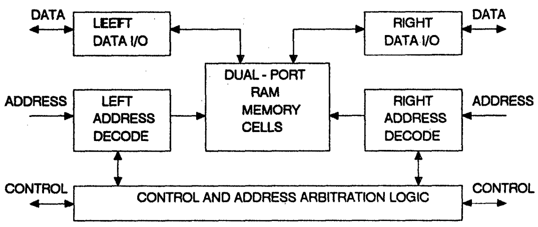


Figure 3. Dual-Port RAM Block Diagram

Data Pins

The data pins are designated I/O0 through I/O7, where I/O0 is the LSB and I/O7 is the MSB. The data pins are bidirectional; their states represent either the data to be written or the data to be read.

Control Pins and Flags

The control pins are Chip Enable($\overline{CE}$), Read/Write (R/$\overline{W}$), and Output Enable ($\overline{OE}$). Two flags are also provided, $\overline{INT}$ and $\overline{BUSY}$; both have open-drain outputs and require external pullup resistors. A LOW level on the chip enable input allows that port to become functional. Data is either read from the internal dual-port RAM array, or written into it, depending upon the state of the Read/Write signal: a LOW initiates a write operation. The three-state, data output drivers are enabled by the LOW state of the output enable signal.

Table 1. The Cypress Dual-Port RAM Family

Cofiguration	Part Number	M/S	Package	Options			
			48-pin Dual	In-line Pkg.	48-pin Square	52-pin	Square
			Ceramic	Plastic	LCC	LCC	PLCC
1KX8	CY7C130	M	D26	P25	L68	---	---
	CY7C131	M	---	---	---	L69	J69
	CY7C140	S	D26	P25	L68	---	---
	CY7C141	S	---	---	---	L69	J69
2KX8	CY7C132	M	D26	P25	L68	---	---
	CY7C136	M	---	---	---	L69	J69
	CY7C142	S	D26	P25	L68	---	---
	CY7C146	S	---	---	---	L69	J69

Note: The Interrupt function is not available at the 2KX8 level in a 48-pin package

When one port writes to a pre-determined memory location (mailbox), an interrupt to the other port is generated. When the interrupted port reads that memory location, the interrupt is reset.

When both ports address the same memory location and both chip enables are active (LOW), there is said to be contention for that address.

When contention occurs, an arbitration is performed, and ownership of the resource (a memory location) is assigned to the winner. The loser of the arbitration is so notified by the active (LOW) state of the busy signal to it.

Detailed Functional Description

The following paragraphs describe the functional operation of the Cypress dual port RAM family.

Interrupt Logic

A simplified functional logic diagram of the interrupt logic of the dual-port RAMs is shown in *Figure 4.* The chip enable signals to this logic are not shown. The chip enable must be asserted for the port to either read from or write to any location, including the mailboxes. Note that the mailbox locations may be used as conventional memory by simply not connecting the interrupt line to the appropriate processor.

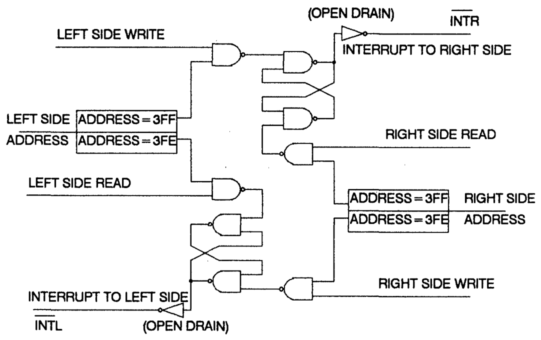


Figure 4. Interrupt Logic

The upper two memory locations (7FF, 7FE for 2K x 8, 3FF, 3FE for 1K x 8) may be used for message passing. The interrupt (request) to the right processor, INTR, goes LOW when the left processor writes to the highest memory location, which is the mailbox for the right processor. When the right processor reads the highest memory location, the flip-flop is reset, and INTR goes HIGH. The interrupt (request) to the left processor, INTL, goes LOW when the right processor writes to the second highest memory location, which is the mailbox for the left processor. When the left processor reads the second highest memory location, the flip-flop is reset, and INTL goes HIGH. Note that each port may read the other port's mailbox without resetting the associated flip-flop.

Interrupt Interaction With Busy

The active state of the busy signal to a port prevents it from setting the interrupt to the winning port. Also, the active state of the busy signal to a port prevents that port from resetting its own interrupt. These operations are ramifications of the concept of the ownership of data.

Functional Operation With Contention; Master

If both ports address the same memory location at the same time there is said to be contention for that address. The master will perform an arbitration and one port will win and the other port will lose. Since there are two ports and each can be in one of two states (either reading or writing), as shown in *Table 2,* there are four combinations of ports and states.

Case 1: Both Ports Reading

If both ports read the same location at the same time, one would assume that there is no problem; both ports should read the same data. This is true for all dual-port integrated circuits. However, under this condition, which is called contention, an arbitration is performed (by the master) and the memory location is assigned to the winner. The loser is not prevented from reading but is notified that the memory location "is owned by" the other port by the active state of its busy signal.

Table 2. Functional Operation of Dual Port Masters

	OPERATION		RESULT OF OPERATION AFTER	ARBITRATION (MASTER)
CASE	LEFT PORT	RIGHT PORT	AMD	CYPTESS and IDT
1	READ	READ	BOTH PORTS READ	BOTH PORTS READ
2	READ	WRITE	LOSER WRITES, WINNER IF READING, MAY HAVE CORRUPTED DATA AND NOT KNOW IT	LOSER PREVENTED FROM WRITING. IF LOSER IS READING AND PORTS ARE ASYNCHRONUS, DATA READ MAY NOT BE VALID
3	WRITE	READ		
4	WRITE	WRITE	BOTH PORTS WRITE, WINNER'S DATA MAY BE CORRUPTED	WINNER WRITES, LOSER PREVENTED FROM WRITING

The Ownership of Data

In order to guarantee data integrity in a multiprocessor system it is standard practice to apply the concept of "the ownership of data". This ownership may be applied to executable code, data, or control locations in memory. The control locations in memory may be associated with a resource (e.g., printer, tape drive, disk drive, communications port).

When arbitration occurs as a result of contention in a Cypress dual-port RAM, one port (the winner of the arbitration) is given temporary ownership of the memory location. The losing port is allowed to read the memory location but is told that it lost the arbitration by the active state of its busy signal.

Cases 2 and 3: One Port Reading and the Other Port Writing

In the AMD dual port the losing port is not prevented from writing. In the Cypress and IDT dual ports the losing port is prevented from writing. All dual ports assert busy to the losing port, so that it can tell that the data may be corrupted.

In the Cypress dual ports the losing port is prevented from writing so that the data cannot be corrupted. Busy is asserted to the losing port, so that it can tell that its read or write operation may not have been successful.

Case 4: Both Ports Writing

In the AMD dual-ports both are allowed to write. Busy is asserted to the losing port, so that it can tell that the data may be corrupted. However, the winning port is not told that the data it just wrote may be corrupted by the writing of the losing port, which could cause system errors.

In the Cypress and IDT dual-ports, the losing port is prevented from writing so that the data cannot be corrupted. Busy is asserted to the losing port, so that it can tell that its write operation was unsuccessful.

Arbitration Logic

A block diagram of the arbitration logic used in the Cypress dual port RAM masters is presented in *Figure 5*. The functions of the arbitration logic are (1) to decide which port will win and which will lose if the addresses are equal simultaneously, (2) to prevent the losing port from writing, and (3) to provide a busy signal to the losing port.

The arbitration logic consists of left and right address equality comparators (with their associated delay [d] buffers), the arbitration latch formed by the cross-

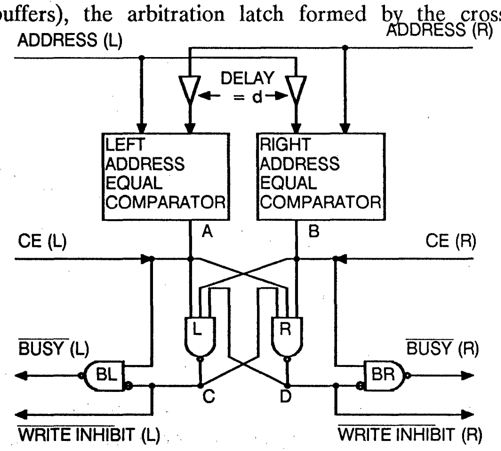


Figure 5. Arbitration Logic

coupled, three-input, NAND gates labeled L and R, and the gates that generate the busy signals.

Operation With Unequal Addresses

When the addresses of the right and left ports are not equal, the outputs of the address comparators (nodes A and B) are both LOW, the outputs of the gates labeled L and R (nodes C and D) are both HIGH, which forces both busy not signals HIGH and both write inhibit not signals HIGH. The arbitration latch does not function as a latch.

Operation With Left Port - Camped on an Address

Next, consider the condition where the left port address and chip enable signals are quiescent and the right port address changes from an address different from that of the left port to an address equal to that of the left port. Nodes A and B are initially LOW.

The right-port address signals do not go through the delay buffer (d), so the output of the right-address comparator (node B) will go HIGH an amount of time, d, before node A goes HIGH. The delay, d, must be greater than the delay through the R gate, so that when node B goes HIGH, node D will go LOW, causing node C to remain HIGH. The CE(R) and CE(L) signals are both HIGH; they are the inverse of the chip enable device inputs. Node D going LOW causes the output of the BR gate to go LOW, which tells the right port that the memory location it just addressed belongs to the left port. A write-inhibit signal is also generated that prevents the right port from writing into the addressed memory location. To summarize; when the right port addresses a memory location that is already being addressed by the left port, after an amount of time equal to the sum of the propagation delays of the right address comparator, the R gate, the BR gate, and the output driver (not shown), the busy signal to the right port is asserted. Nodes A, B, and C are now HIGH and node D is LOW. $\overline{\text{BUSY}}$ is asserted to the right port.

Operation With Right Port - Camped on an Address

A little thought should convince the reader that, due to the symmetry of the arbitration logic, the operation of the circuits, when the right port camps on an address and the left port then addresses the same location, is similar to that of the preceding paragraph.

Both Right and Left Port Addresses Equal Simultaneously

In the general case, unless guaranteed not to occur by the design of the system, it is possible to have both ports access the same memory location simultaneously. When nodes A and B go from LOW to HIGH at exactly the same instant, the arbitration latch will settle into one of two states and will determine which port wins and which port loses. The latch is designed such that its two outputs will never be LOW at the same time. It is also designed to have a very fast switching time.

Port Setup for Priority

There is a minimum time difference between either, (1) the two chip enables going from inactive to active, or (2) the two sets of addresses going from mis-match to equal, after which the first port is guaranteed of winning the arbitration. This parameter is called "port set-up time for priority", and is abbreviated as t_{PS} on the data sheets. The specified value is five nanoseconds. Cypress product engineering has measured this parameter at room temperature and nominal Vcc (5 Volts) and determined it to be approximately 200 picoseconds. This same parameter could be specified between the addresses of one port and the chip enable of the other, but it is not.

Other Busy Key Parameters

There are several other key parameters that are specified with respect to the busy signal. Their significance will be explained and the ramifications of their values will be analyzed.

Busy LOW from address match, t_{BLA}, is the maximum time it takes busy to go LOW, as measured from the time the two port addresses are the same. This is the time from an address match until the losing port is notified that it has lost the arbitration. Obviously, the sooner this occurs the better. If the value of t_{BLA} is greater than the the memory cycle time it means that another cycle must be added in order to detect the condition, which may severely reduce performance. This time is less than the minimum cycle time for all speed grades of all Cypress dual-port RAMs.

$\overline{\text{Busy}}$ HIGH from address mismatch, t_{BHA}, is the maximum time it takes busy to go from LOW to HIGH, as measured from the time the two port addresses do not match until the busy signal goes HIGH. The comments of the preceding paragraph apply.

The next two parameters are similar to the preceding two. The difference is that the chip enable controls the busy signal. They are $\overline{\text{BUSY}}$ LOW from $\overline{\text{CE}}$ LOW; t_{BLC}, and $\overline{\text{BUSY}}$ HIGH from $\overline{\text{CE}}$ HIGH; t_{BHC}. Both of these parameters are less than the minimum cycle time for all speed grades of all Cypress dual-port RAMs.

$\overline{\text{Busy}}$ HIGH to valid data, t_{BDD}, is the maximum time it takes the data to become valid to the losing port after $\overline{\text{BUSY}}$ goes away. The value of this parameter is equal to the address access time, t_{AA}, because a read cycle is initiated to the losing port when its $\overline{\text{BUSY}}$ signal transitions from LOW to HIGH. The cause of the busy transition could be an action by either port. The winning port could either change its address or deassert its chip enable signal. Refer to the following paragraph for the conditions that will trigger a valid read to a port that has just lost an arbitration to a port that is writing.

The final two parameters are shown in *Figure 6*, which illustrates the timing for the right port performing a write operation, and the left port asynchronously moving to the same address and attempting to perform a read operation. The first parameter of interest is t_{DDD}, which is the maximum time from when the data to be written by the winning port is stable until that same data is valid at the outputs of the port that received the busy. The second parameter of interest is t_{WDD}, which is the maximum time from the write strobe HIGH to LOW transition of the winning port until the data is valid at the outputs of the port that received the busy.

In the general case, when the two ports are operating asynchronously (i.e., independent clocks), and the conditions illustrated in *Figure 6* occur (winning port writing and losing port reading), it is possible for the losing port to read either the old data, the new data, or some random combination of the two. If the read occurs early with respect to the write, old data will be read. If it occurs late with respect to the write, new data will be read. And, if the read occurs at the same time that the data is changing from old to new, the data read will not be predictable. However, all is not lost. There are two general solutions. Both use the fact that the busy signal is asserted to the losing port, telling it in this instance that the data it is reading may not be valid.

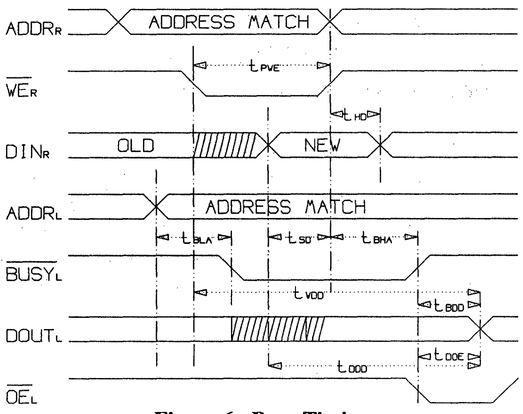


Figure 6. Busy Timing

One solution is to use the HIGH to LOW transition of the busy signal to the losing port to generate an interrupt to the processor (or state machine) so that it can repeat the operation. The drawback of this technique is that a "snapshot" of the states of the address lines and the read/write line of the losing port must be taken, so that the processor can tell what load/store operation caused the interrupt. This requires latches or flip-flops for the data, control logic for doing the sampling, and uses up an interrupt line. The processor must also be able to read the sampled data later.

A second solution is to use the LOW level of the $\overline{\text{BUSY}}$ signal to the losing port to either delay the reading of data until it becomes valid, which occurs an access time after the LOW to HIGH transition of busy, or insert "wait states" until busy goes HIGH, or stretch the clock until busy goes HIGH. This will probably require less hardware and control logic than the preceding approach. It does mean that the busy signal must eventually go from LOW to HIGH. This will happen when the winning port either changes its address or deasserts its chip enable. For this reason, as well as system noise immunity and power saving considerations, it is recommended that "blocks of addresses" be decoded in order to generate chip enables for the dual ports. However, the losing port has no control over the winning port in the general case, so the question is, what can the losing port do in order to successfully read the data just written (assuming the winning port does not change its address, write, or chip enable signals) ?

The answer is, there are two operations the losing port can perform in order to initiate a successful read operation from an address that is temporairly owned by a winning port. They are:

1. Change an address line to a different address and then change it back to the original address. This toggles the busy signal to the losing port.

2. Change the state of the chip enable signal. This also toggles the busy signal to the losing port.

These two operations by a losing port or changing either any address line or the chip-enable line by the winning port will initiate a successful read to the losing port when the winning port is writing. The next question to be answered is why ?

ATD (Address Transition Detection)

The reason is that all Cypress dual-port RAMs, both masters and slaves, use a circuit design technique called Address Transition Detection (ATD) in order to improve performance to and reduce power dissipation.

ATD Improves Performance

Performance is improved by equilibration of differential paths, pre-charging critical nodes, and forcing the outputs to a high-impedance state. Equilibration and pre-charging bias critical nodes to voltage levels approximately in the mid-point of the small-signal operating range, so that when the data is sensed it takes a shorter amount of time to transition to the ZERO or to the ONE level. Forcing the outputs to their high-impedance states improves speed slightly, but more importantly, reduces output switching noise by eliminating crowbar current and separating the output current into two pulses instead of one.

ATD Reduces Power Dissipation

Power dissipation is reduced by turning on power-hungry circuits only when they are required. Slightly over fifty percent of the circuits in a RAM are linear and approximately seventy percent of the power is dissipated in the sense amplifiers during a read operation. When the RAM is operating at its maximum frequency the ATD circuits are being constantly triggered, so the power savings are minimal. However, at lower speeds or smaller duty cycles the power savings are significant. A diagram representing a typical ATD sequence is illustrated in *Figure 7.* The event that triggers the ATD sequence (per port) is the transition of any address signal, the chip enable signal, or the read/write signal. Equilibration and pre-charging are performed next, followed by either turning on the sense amplifiers and latching the data (read operation) or pulling the BIT and $\overline{\text{BIT}}$ lines to the required levels (write operation) at the addressed location. The master clock pulse lasts from seven to eleven nanoseconds, depending upon temperature, supply voltage, and the distributions of integrated circuit processing parameters. At the end of the pulse the data is latched and the appropriate circuits are turned off.

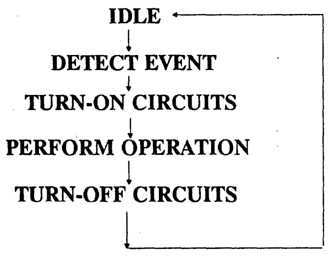


Figure 7. Simplified ATD Sequence

Standalone Operation of the Master Dual-Port RAM

A block diagram showing the interconnections between the major elements in a system using two eight-bit microprocessors, the Cypress CY7C132 dual-port RAM, static RAM, and EPROM is presented in *Figure 8.*

The key points are (1) the address lines of each microprocessor are decoded to generate the chip enable signals to the dual port, the RAM, and the EPROM, (2) the interrupt requests to the microprocessors require pullup resistors, and (3) the busy signals, which go to the wait inputs of the microprocessors, also require pullups.

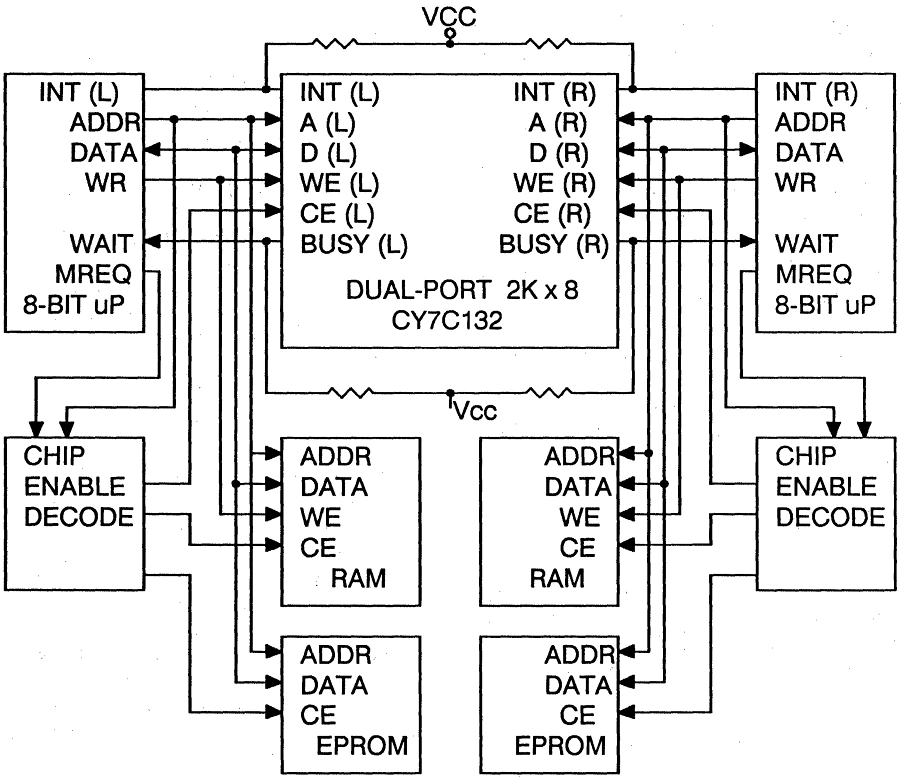


Figure 8. Typical 8-Bit Microprocessor

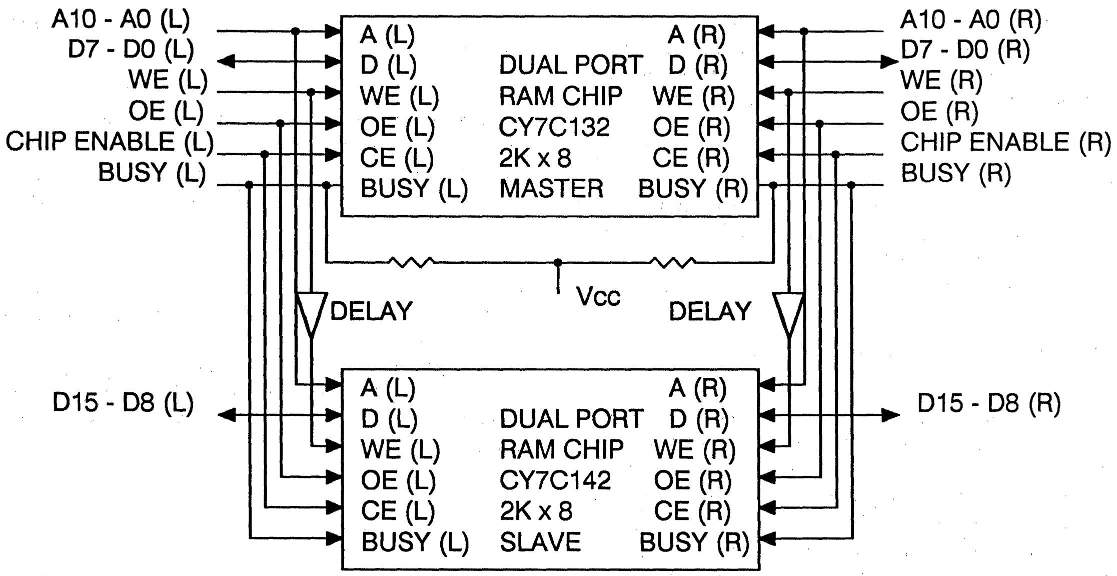


Figure 9. Expansion (2K x 16) With Slave

Word Width Expansion Using the Slave

A block diagram showing how to interconnect a CY7C132 (2K x 8) master and a CY7C142 (2K x 8) slave together to form a 16-bit wide word is presented in *Figure 9.* The interfaces to the processors are not shown. Also not shown are the connections for the interrupt signals. The interrupt outputs are not available at the 2K X 8 level in the 48-pin DIP due to pin limitations. In the LCC and PLCC packages the interrupt outputs are available from both the master and the slave devices. Either one may be used. It is not required to tie the corresponding interrupt pins of the master and the slave together.

Delaying of the Write Strobe

In width expansion, the write signals to the slave devices must be delayed an amount of time at least equal to t_{BLA}, which is the time required for the master to assert the busy signal to the slave after an address match. These delay elements are labeled DELAY in the diagram. This prevents the data in the slave at the address in contention from being overwritten. The write cycle time must be increased by this amount of time. In equation form;

$$t_{WC} = t_{PWE} + t_{BLA}. \qquad \text{Equation 2}$$

Where the delay of the delay element must be at least equal to t_{BLA}.

Note that if more slaves are added to make a wider word, (e.g., 24-bits, 32-bits) the outputs of the delay elements can be connected directly to the write strobe inputs. Additional delay elements are not required.

Standalone Operation of the Slave

Certain applications may require giving one port permanent and absolute priority over the other. This can easily be done by implementing the memory using only slave dual-port RAMs. The $\overline{\text{BUSY}}$ input to the priority port must be tied HIGH by either connecting it directly to Vcc, or to Vcc through a 10 K Ohm pullup resistor. The $\overline{\text{BUSY}}$ input of the low priority port may be connected to the read/write input of the high priority port.

In this configuration, the busy (read/write) signal to the lower priority port will always prevent it from writing when the high priority port is writing (to any and all locations). In the general case, when the two ports are operated asynchronously, and the lower priority port is writing, and the higher priority port simultaneously writes, the data of the lower priority port will be overwritten. Admittedly, this is not a very elegant solution because the $\overline{\text{BUSY}}$ input to the low priority port is not qualified by comparing the addresses of the two ports or their chip enables. However, it stimulates the reader to think of how the slave dual-port RAMs can be used with external arbitration logic. The busy inputs may be used by either control logic or under program control to dynamically change the priority of the ports.

If the lower priority port is "read only" its $\overline{\text{BUSY}}$ input may be tied HIGH by either connecting it directly to Vcc, or to Vcc through a pullup resistor.

Dual-Port Design Example

The following design example is presented to illustrate the methodology to follow when designing with Cypress dual-port RAMs.

Design Specification

A dual port memory is to be used for message passing and bus snooping between many bus masters on a 32-bit wide system bus on one side (the right side) and a 16-bit processor on the other side (the left side).

From the right port the memory appears as 8K 32-bit words and from the left port it appears as 16K 16-bit words.

Design such a memory with the following additional constraints:

1. The memory location corresponding to address zero for both ports is the same.

2. The data read from and written to the memory from both ports is in the same order (i.e., D0 of the right port corresponds to D0 of the left port). Also, D16 of the right port appears as D0 of the left port in address location 2048.

3. The minimum cycle time is 35 nanoseconds.

4. In order to conserve power, blocks of addresses will be decoded to generate the required chip selects.

5 The CY7C132 and CY7C142 dual-port RAMs are to be used.
Part of the design is to chose how many masters and how many slaves are required and how they are to be interconnected.

6. The appropriate $\overline{\text{BUSY}}$ signals are to be generated to the correct port when contention occurs.

7. All possible mailbox locations that can be used for message passing are to be used. Explain how the interrupt lines should be connected.

8. Name the right port signals AR0 ...AR12, DR0...DR31, $\overline{\text{CER}}$, and $\overline{\text{BUSYR}}$. Name the left port signals AL0...AL13, DL0...DL15, $\overline{\text{CEL}}$, and $\overline{\text{BUSYL}}$.

Show a logic diagram, interconnections, and timing.

Design Overview

A simplified logic diagram of the memory is presented in *Figure 10.* A total of sixteen 2K X 8 dual-port RAMs are required. The devices labeled MA (Master, bank A) through MD (Master, bank D) are CY7C132 masters and the devices labeled SU (Slave, Upper half-word) and SL (slave, Lower half-word) are CY7C142 slaves. The memory consists of four masters and twelve slaves, together with the required control logic.

From the right port the memory is configured as 8K 32-bit words, with a master controlling three slaves. The one of four decoder labeled RB (Right Bank) generates chip enable signals for each bank of 2K 32-bit words. In this example data is written (sampled) on the bus side and the only reads performed are from the mailbox locations. A general purpose "right port control logic" block is shown that generates control signals which conform to the timing diagram of *Figure 11.* The generation of the output enable control signals is not illustrated, but they are similar to the RB decoder. If the application does not require message passing to the right port, the right port output enable pins of all of the dual port RAMs may be tied directly to Vcc.

From the left port the memory is configured as 16K 16-bit words. For this organization the reader is probably thinking that the slave dual ports in the second column from the right in *Figure 10* should be masters. However, if this were done, the arbitration logic in them would have to be defeated when the right port addressed the same address, which would add logic, reduce the speed, and complicate the design. Therefore, a combination of left bank decoding (LB, 1 of 4 decoder) and upper-lower (UL, 1 of 8 decoder) 16-bit word decoding is used to cause the "bank master" to arbitrate when the left port is addressing the same bank as the right port.

Operation of the Right Port

For purposes of this discussion, the word word refers to the 32-bit long word at the right port system bus interface. At the 16-bit processor interface the 32-bit word will be refered to as either the lower half-word (bits zero through fifteen of the right port), or as the upper half-word (bits sixteen through thirty-one of the right port).

Bank Selection Using the Chip Enables

The one-of-four RB decoder decodes the four combinations of the right port upper two address bus signals and generates four active-low chip enables to each bank of four dual port RAMs. Bank A contains addresses zero through 2047, bank B contains addresses 2048 through 4095, bank C contains addresses 4096 through 6143, and bank D contains addresses 6144 through 8191. In other words, bank A addresses 0 to 2K, bank B 2K to 4K, bank C 4K to 6K, and bank D 6K to 8K.

Addressing and Write Strobe

The lower eleven right port address lines, AR(0:10), are connected, respectively, to the A0 through A10 right port address pins of all of the dual ports.

The generation of the write strobe is not shown, but the timing is illustrated in *Figure 11.* Note that the signal is applied directly to all of the masters in parallel, then buffered, and then applied to all of the slaves. The minimum propagation delay of the buffer must be at least as large as t_{BLA}, which is the time required for the master to assert the busy signal to the slaves after an address match occurs.

Note that all of the right port output enable pins are connected together and should either be driven if reading is required, or connected to Vcc if not.

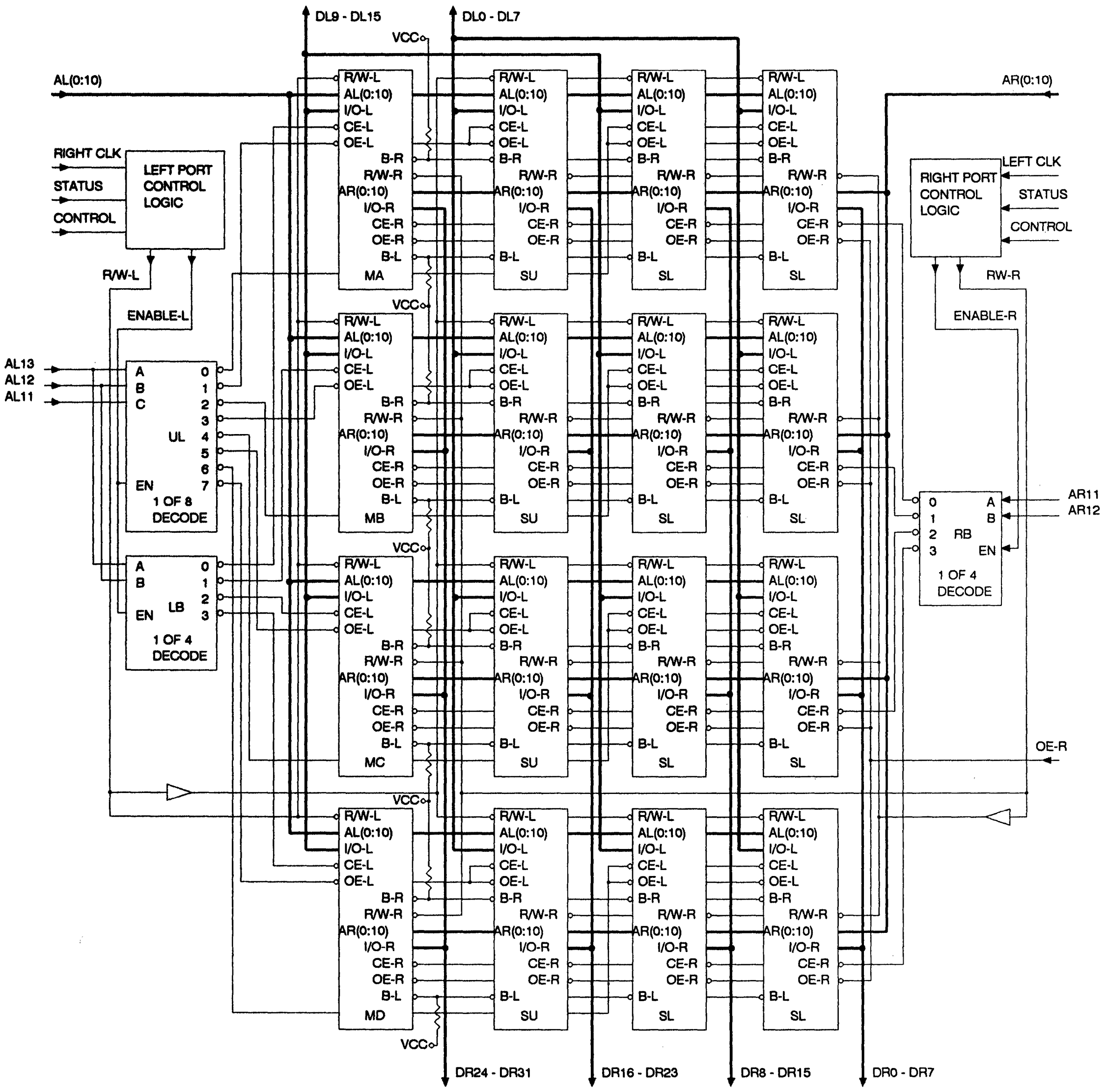


Figure 10. Logic Diagram for Dual-Port Example

Right Port Busy Connections

The open-drain busy outputs of the right port masters must be pulled up to Vcc using resistors. A value of 330 Ohms is recommended. The master busy output is then connected to all of the right port slave busy inputs for each bank.

Data Bus Connections

The I/O pins of each column of RAMs are connected to their respective I/O pin on each bank. This "OR-tie" connection is allowed because the bank selection chip enable causes the output buffers of the un-selected banks to go to the high-impedance state.

Operation of the Left Port

Bank selection is performed by the one-of-four decoder labeled LB. The upper two left port address lines, AL13 and AL12, are used to decode bank select chip enable signals for the four masters only. Bank A corresponds to addresses 0 through 4095, bank B corresponds to addresses 4096 through 8191, bank C corresponds to addresses 8192 through 12,287, and bank D corresponds to addresses 12,288 through 16,383.

Upper and Lower Half-word Selection

The one-of-eight decoder labeled UL decodes the upper three right port address signals to generate eight chip enable signals with a resolution of 2048. The respective chip enables are applied to the chip enable and output enable pins of the slaves (2048 resolution) and to the output enable of the masters. Because the master chip enable resolution is 4096 it arbitrates for two blocks of 2048 16-bit half-words.

Addressing and Write Strobe

The lower eleven left port address lines , AL(0:10), are connected, respectively, to the A0 through A10 left port address pins of all of the dual ports.

At the 16-bit interface, writing is required only if the left port wishes to send a message to the right port. Otherwise, the left port write pins of all of the dual ports may be connected to Vcc.

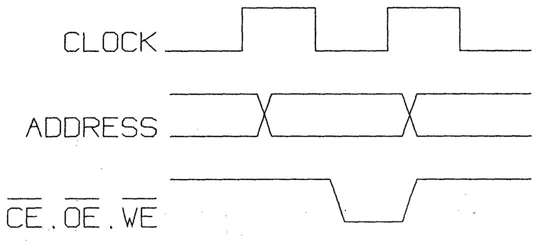


Figure 11. Dual-Port Timing for Example

Left Port Data Bus Connections

The respective data I/O pins of the left port are connected together in the same manner as those of the right port for all RAMs in the same column. In addition, in order to multiplex a 32-bit wide data word to a 16-bit half-word, the least significant bytes and the most significant bytes of each 2048 word group are connected together. The UL decoder that controls the output enable of the left port performs the selection.

Interrupts (not shown).

The interrupt pins of the masters (if used) should be pulled up to Vcc through a 330 Ohm resistor and connected to the processor interrupt request input. The interrupt pins of the slaves may be left unconnected.

Control Signal Terminations

If the control signal connections from their source to the dual port memory are "long lines" they may require proper termination in order to avoid voltage reflections due to impedance mis-matches. Please refer to the applications paper titled "Systems Design Considerations When Using Cypress CMOS Circuits" in this handbook for further information.

References

1. Dijkstra, E.W., "Solution of a Problem in Concurrent Programming Control". CACM Vol 8, no.9, Sept. 1965, p569.

2. Dijkstra, E.W., "Co-operating Sequential Processes". Programming Languages, F. Genyus (Ed.) Academic Press, New York, 1968, pp43...112.

CYPRESS
SEMICONDUCTOR

Using Dual-Port RAMs Without Arbitration

Dual-Port Applications

One of the most common applications for dual-port RAMs is to provide a high-speed shared memory resource between two processors in a system. *Figure 1.* illustrates how the two processors communicate by passing data and commands via the shared memory. Both processors benefit by having access to the dual-port since it is mapped just like any other memory device on the board. Fast, local access to the shared memory eliminates the need to arbitrate for and access the system bus in order to read or write a common resource area such as a shared memory card. In fact, many multiprocessor embedded control systems implement dual-ports for interprocessor communication and choose to eleminate the system bus entirely. Removing the burden of a system bus, which only exists to "hook" the processors together, reduces not only the complexity of the system, but the part count and power consumption as well.

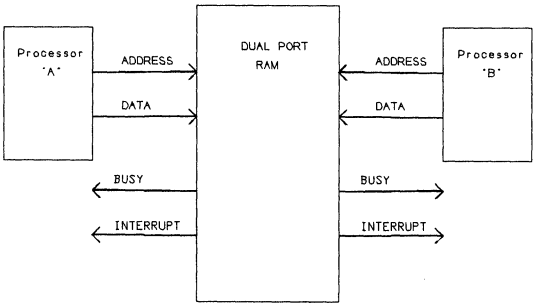


Figure 1. Dual-Processor Communication

Incorporating dual-ports into a design, such as the dual processor example, is very straight forward. It is important however to consider the case of an address contention or "busy" situation that can arise when both processors simultaneously attempt to access the exact same location. Cypress dual-ports have several mechanisms which simplify these simultaneous access conditions. The simplest approach to resolving contention is to use the dual-port's "BUSY" output lines. Both right and left ports provide a busy output signal. Busy is activated by the arbitration logic inside the dual-port when it senses a match between the left and right address lines. Assertion of busy indicates that both ports have attempted to access the same location in the RAM. In the case of a dual processor system, these signals can easily be gated with the processor's local WAIT signal in order to generate a hold to the micro until the busy is deasserted. Adding an occasional wait state to a microprocessor generally has no effect on the overall system performance. Gating the wait line and generating a hold to the processor resolves the logical problem of simultaneous address conflicts, but does not address the system level issues that can cause it. For example, in the case of two microsystems, we can see a common underlying cause of a busy state. In this example processor A attempts to read an array of data that was generated by processor B. However, there is no mechanism to alert processor "A" when the data is really ready or valid. Therefore, the system potentially runs into the situation where processor "A" is updating a RAM location while another processor, "B", is reading the same address or vice versa. This lack of overall synchronization or interprocessor communication can manifest itself as stale data or incomplete arrays of data in the shared memory. In a few cases, stale or incomplete date is tolerable, but in most it can be fatal. Locking a processor or processors out of certain areas of memory until the data is available will guarantee that the processors never receive stale data. In order to implement address space restrictions, it is necessary to provide a higher level of access protection above the basic gating of busy technique. In most cases, it becomes necessary to add some external hardware which "signals" the processors that new data is available or that it now has permission to access a certain area of the dual-ported device. Interrupts serve well as a simple means of alerting or synchronizing interdependent system elements that pass data via a shared memory. Cypress dual-ports provide interrupt lines, as outputs, in order to simplify the task of interrupting or signaling the processors that relieves the designer of the need to create his own interrupt mechanism. Assertion and deassertion of these interrupt lines is accomplished by performing write and read operations to special locations within the dual-port. The read and write operations are listed in *Table 1* below.

Table 1. Interrupt Line Usage

Function	Result
Write to left Address 3FFh	Asserts Int_right
Read from Right Address 3FFh	Removes Int_right
Write to Right Address 3FEh	Asserts Int_left
Read from Left Address 3FEh	Removes Int_left

The data word written to, 3FEh and 3FFh, can be used as a status word. This data word is presented to the data bus during the read operation of an interrupt removal cycle. This status word, or semaphore provides additional "system level" information that augments the hardware interrupt signal by adding the ability to easily pass along some meaning with the actual interrupt event. More simply, the interrupt line alerts the processor that some action is required and status word provides additional information as to exactly what happened or what needs to be done. The actual meaning of this status byte that is passed between processors is defined by the system designer. Generally, the status byte is used to indicate that data is ready, to lock a processor out of a specific range of addresses, or to prompt a processor for new data. Using the interrupt, along with status information, is an easy way of avoiding busy conditions by synchronizing processes or restricting address spaces via software. The designer now has several options for dealing with simultaneous address situations. Busy can be used in a strictly hardware solution, or interrupts coupled with status words may be chosen for a software solution. Regardless of the designers preference for a hardware or software approach, Cypress dual-ports provide all signals and functions necessary to insure a simple and effective system solution that maintains data integrity and system sanity.

Using Dual-ports Without Arbitration

Wait states and interrupts are a good solution for systems with microprocessor-like elements that are not affected by an occasional wait state. However, there is another much broader class of systems and applications that cannot tolerate any type of data flow interruption or busy condition. Typically, these systems are "dedicated function units" that are rigidly pipelined and operate on continuous or nearly continuous streams of data. A high-speed video processor is a good example of a system whose elements cannot be wait stated due to the constraint that requires a data word or pixel be processed in every clock cycle. The block diagram in *Figure 2* shows a video data transform or look-up table.

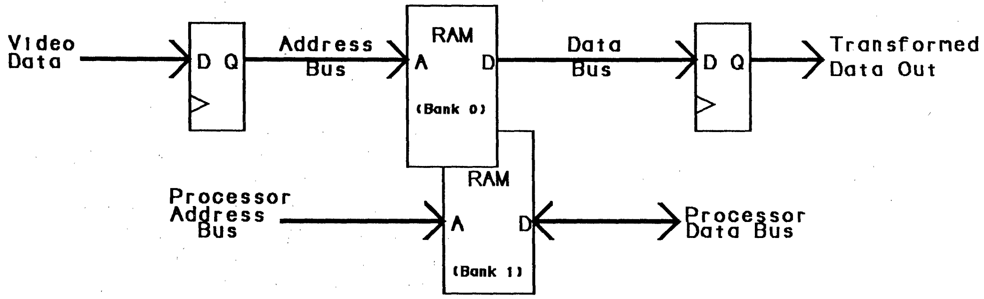


Figure 2. Video Look Up Table

This implementation uses a very common dual-banked, or "ping-pong" RAM to realize a look up table translation function as seen in *Figure 3*. A continuous stream of video data drives the address lines of RAM bank 0. The output or transformed data of bank 0 then flows downstream to the post processor units. Meanwhile, as continuous video data is flowing through RAM bank 0, the transform table of second bank, bank 1, is updated by a processor element without interfering with the video data flow. Dual banks make it impossible for a busy condition or address conflict to exist, since each system element essentially has its own discrete dedicated RAM. When the processor has completed its task of updating the look-up table, it swaps RAM banks by toggling the Bank-select line. The PAL then changes the state of the buffer-enable signals, which redirects the data flow of the two banks of RAM. The ping-pong arrangement is effective, however, the implementation is

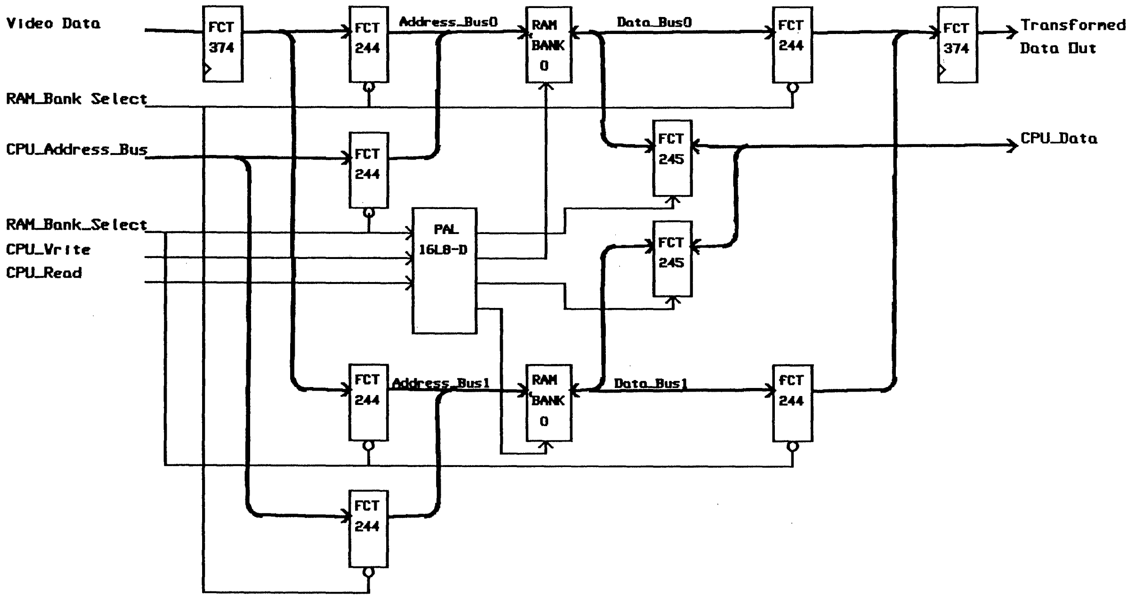


Figure 3. Ping-Pong RAM Array

very costly in terms of real estate. At least eleven very high speed devices are required to build this function using standard static RAMs. Replacing the buffers, logic, and SRAMs with a single dual-port RAM (*Figure 4*) in this application would simplify the design substantially. Video data utilizes the left port of the device, while the processor communicates with the right port.

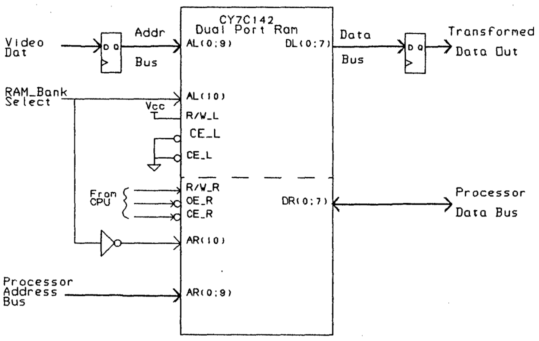


Figure 4. Video Lookup with Segmented Dual-Port RAM

Having two ports eliminates the need for any type of data and address steering buffers. There still remains, however, the problem of simultaneous address accesses and busy conditions that could potentially exist during processor update cycles. RAM segmentation eliminates the possibility of a busy conflict and is the key to implementing a dual banked RAM within a single dual-port. Segmentation is accomplished by the use of a single inverter. The Bank_select signal from the processor drives the left address port MSB and its inverse drives the right MSB. The dual-port has now been separated or segmented into two 1K addresses spaces that do not overlap. The dual-port now appears a two totally separate RAMs just as it did in the ping-pong implementation. Again, since the left address can never equal the right address due to the opposite state of their MSB's a busy condition is not possible. Choosing to use a dual-port does more than simplify the design. *Table 2* clearly shows the tremendous saving in real estate and power consumption.

Table 2 demonstrates that a single dual-port device reduces the board area by 68% and reduces the power consumption by almost 80%. System reliability in terms of MTBF benefits greatly by having fewer components and significantly lower power dissipation. The multitude of buffers and transceivers that steer data and ad-

Table 2. Dual-Port v.s. Ping-Pong RAM Comparison

Device	Qty	Power (Ma)	Size (Sq.in.)
FCT244	6	15	0.4
FCT245	2	10	0.4
PAL16L8-D	1	180	0.4
20nsx8 RAM	2	140	0.52
Total	11	570	4.64
CY7C142-35	1	120	1.5

dress signals in a ping pong memory array not only take up relatively large amounts of board space but also add to the prop delay of the data forcing the designer to use very high-speed RAMs. Dual-ports do not suffer from the added burden of buffer delays and, therefore, can operate at significantly lower speeds.

There are many types of high-speed data-processing applications that can benefit form the use of dual-ports. For example, high speed video or radar data is often transmitted in a nonsequential or cross interleaved order. The receiver must first descramble or reorder the data before it can be used. Again, the incoming data stream cannot be stopped in the event of an address contention. *Figure 5* shows that a dual-port is again an ideal solution for this type of problem. Incoming data is written into the left port of the RAM in the order that it was received. The pixel counter provides sequential addresses to the left side of the dual-port

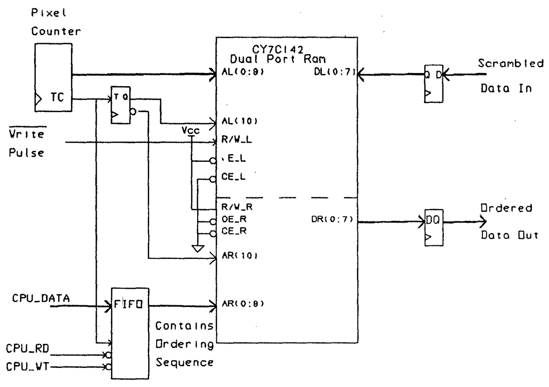


Figure 5. Data Descrambler

and increments after each pixel. At the end of the first line, the counter reaches terminal count and initiates a bank toggle via a "T" type flip-flop function. After the banks switch, the new data is accessible via the right port. A FIFO which is used to store the reordering sequence, drives the right port's address lines in order to read out the stored video data. Notice that by using a FIFO there is no need for counters to generate addresses for the reording sequence table. This function can also be implemented with PROMs and counters but requires more parts and is much less flexible. The CPU is responsible for initializing the descrambling FIFO at boot up. This initialization is only required once since the FIFO utilizes its retransmit function (refer to CY7C429 FIFO data sheet), unless the data ordering changes. We can again ignore the problems caused by address contention because the design implements the dual-port as a segmented memory.

Interfacing a high-speed-pipelined digital signal processor or a bit-slice processor to the system CPU is another very common system interface problem. Coefficients and commands must be passed to the pipelined processor and final results read back by the CPU. Dual banks of RAM are often implemented as a solution because they provide a shared memory space that can be used by both elements of the system without the threat of address contention. Again, since the machines are rigidly pipelined, they cannot easily be stopped or interrupted. Therefore, a single segmented dual-port (*Figure 6*), or several in parallel with no additional glue logic, is a simple cost effective solution to this problem. If two banks of data are too restrictive the dual-port can be segmented into multiple address spaces by simply reserving more or the upper address line pairs. This scheme allows the processor to easily and quickly communicate with the pipeline processor without using large amounts of real estate and power.

In summary, dual-port Static RAMs are a great solution for interprocessor communication problems. By utilizing some simple techniques such as RAM segmentation, the designer can now implement dual-port solutions without regard to arbitration and address contention. These techniques allow the use of dual-ports in a very broad spectrum of applications. Dual-ports are not only extremely versatile and easy to use, but they also add the benefits of simplicity, reduced board space, lower power consumption, and increased reliability.

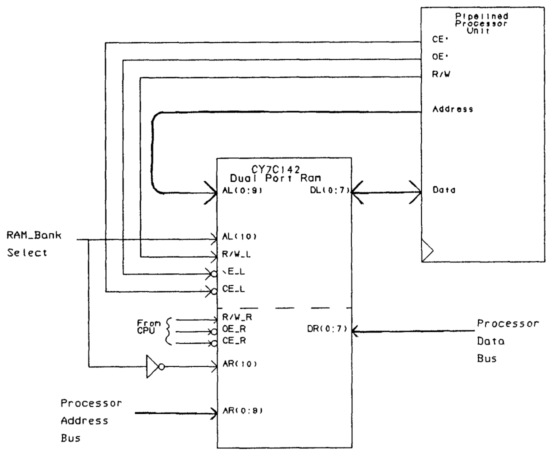


Figure 6. CPU/Pipelined Processor Interface

NOTES:

Section Contents

CYPRESS SEMICONDUCTOR

Memory System Design for the CY7C601 SPARC Processor

Introduction

This paper describes a simple 25 MHz CY7C601 memory design for non-cache memory applications. The memory subsystem consists of 128 KB of data RAM and 128 KB of instruction RAM. (The instruction RAM is easily expandable to 256 KB with the current design.) The distinction made, between data memory and instruction memory for this design, is that the CY7C601 Integer Unit (IU) is not allowed to write instruction memory. This implies that instruction RAM will be loaded at power-up by an external device.

The design will utilize the CY7C157 cache RAM, which has been designed specifically for use with the CY7C601 SPARC Integer Unit (IU) and the CY7C604/605 Cache/Memory Management Unit (CMMU). When used in this environment, all necessary control signals (byte-writes and output enables) are provided by the CMMU. However, this article shows that the CY7C157 can be easily adapted for use in non-cache applications.

First, a description of the CY7C157 will be provided, followed by a brief description of the CY7C601 bus interface. Finally, a design using the CY7C330 EPLD to generate the required byte-write signals will be described. Also, a CY7C332 EPLD design will be described that provides the required output enable signals. A block diagram is shown in *Figure 1.*

CY7C157 Cache-RAM

The CY7C157 cache RAM is a very high-performance 16 K x 16 bit static RAM. This device employs common I/O architecture and a self-timed byte-write mechanism. The self-timed write relieves the designer of the difficult task of generating accurate write strobes in high-speed systems. Address and write-enable inputs are loaded into input registers on the rising edge of the system clock. Data-input and data-output latches are provided, and an asynchronous output enable is also present. The CY7C157 is available in 20, 24, and 33 ns speed grades. A 25 MHz IU requires the slowest device offered, 33 ns, so this device has been chosen for the design.

CY7C601 Bus Interface

The IU has a 32-bit address bus and can directly address four gigabytes of memory. The address bus, data bus, and all memory interface signals (except INULL), are sent "unlatched" in the cycle prior to use and should be latched externally before they are used. Refer to the Bus Cycle sections for more detailed information.

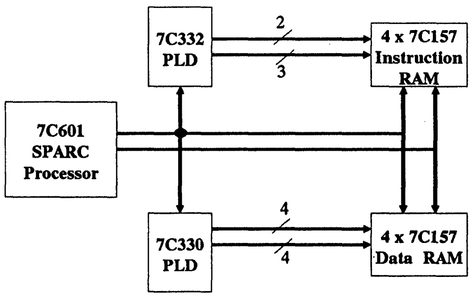


Figure 1. Block Diagram

Memory Wait States and Memory Exceptions

The memory design described does not require wait states, but information on this topic and on memory exceptions is provided in the IU data sheet.

Bus Cycles

If we assume that our system does not contain a floating-point processor or a coprocessor, the bus cycles that must be dealt with are: instruction fetch, load single, load double, store single, store double, and atomic load/store.

Instruction Fetch

Address and control bits are sent out at the beginning of the fetch cycle and must be latched externally. Instruction data is expected at the end of the fetch cycle, when it is then latched from the data bus into the on-chip instruction register. The first cycle in *Figure 2* shows an instruction fetch. All instruction fetches are single-cycle operations, so no pipeline delays are incurred. Under some conditions, the processor is unable to fetch an instruction, usually because a prior multi-cycle instruction needs to use the bus. When this occurs, the processor asserts INULL to indicate that the current fetch cycle should be nullified.

Load Cycles

The first and second clock cycles in *Figure 2* show the timing for a load single integer instruction. Load single integer is a two-cycle operation: logically the first cycle fetches the load instruction, however due to the processor pipeline the instruction is actually faetached at least two cycles prior to the load/store cycle, and the second cycle actually loads the required information from memory. A load double instruction is similar to the load single instruction except that a third cycle is added to fetch the second data word from memory. This is also illustrated in *Figure 2*.

Store Cycles

Figure 3 illustrates store single and store double instructions. A store single requires three clocks. For simplicity assume the store instruction is fetched during the first clock. During the second clock, the destination address of the store is driven onto the bus. Store data is driven onto the data bus at the middle of cycle two and removed at the middle of cycle three. Memory update occurs in cycle three. The early arrival of the store address allows it to be checked for possible write-protect violations or memory exceptions in systems that implement these features.

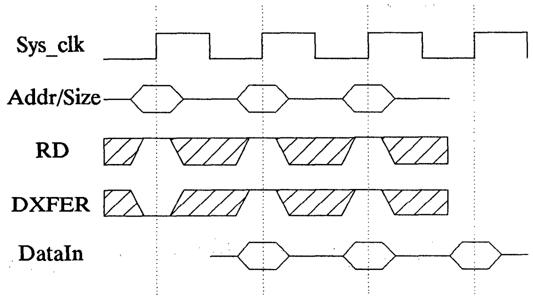
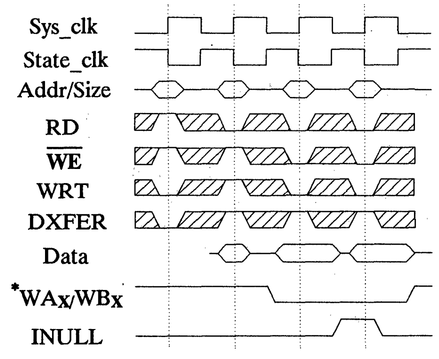


Figure 2. Load/Load Double Timing

The store double instruction is very similar to a store single instruction, except for an extra cycle needed to store the second data word. Note that the address of the second store is set to the first address plus 4, and that the size bits are set to 11, indicating a double-bus access.

Atomic Load/Store Cycles

Atomic transactions consist of two or more transactions which are indivisible; once started, the sequence cannot be interrupted. To ensure bus access for the second



* Signal from 7C330 PLD

Figure 3. Store/Store Double Timing

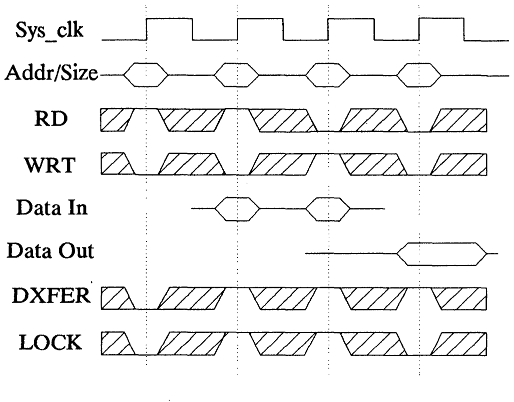


Figure 4. Atomic Load/Store Timing

transaction, the IU asserts LOCK for the necessary length of time. *Figure 4* shows the timing of an atomic load/store instruction.

Design Considerations

Using the CY7C157s in a non-cache application requires generation of appropriate byte-write signals and generation of appropriate output enables. The CY7C157 does not require a chip select when used with the CMMU device, so none is provided. This means that separate sets of write enables must be decoded for each 64 KB (16K word deep) block of RAM. Also, an output enable must be generated on 16K word boundaries during reads. Since address and data setup/hold requirements between the IU and the CY7C157 are guaranteed by design, we will concentrate on the CY7C157-33's write-enable and output-enable timing requirements.

The CY7C157 requires a 6 ns write-enable setup to clock low time and 3 ns write-enable hold from clock low. From the store transaction timing diagrams, it can be seen that the store data valid times are referenced to the falling edge of the system clock, while transaction information (address, size, etc.) is referenced to the rising edge of the same clock. The desired PLD architecture for the write enable generator would provide one clock for clocking in the transaction information and a separate clock for clocking out the write enables. The Cypress CY7C330 state machine is such a PLD. The next critical factor is: Will the CY7C330 meet the write enable setup and hold times? Inspection of the CY7C330-50WC data sheet for tCO and tOH specs shows that these conditions are met. *Figure 5* shows that any write enable is valid 15 ns after the falling edge of Sys_Ck (thus providing a 25 ns setup time) and is held for 3 ns after the falling edge of Sys_Ck (matching the required hold time at the 7C157).

For reads, *Figure 5* shows that the CY7C332 output delay plus the CY7C157 output enable time provides a 5 ns data setup time, which easily meets the 3 ns requirement of the IU. Data hold time requirements are determined by examining the CY7C332 output enable hold time from the Sys_Ck falling edge. This hold time is 3 ns which, when added to a 2 ns minimum turnoff time for the CY7C157, guarantees the required 5 ns data hold time at the IU.

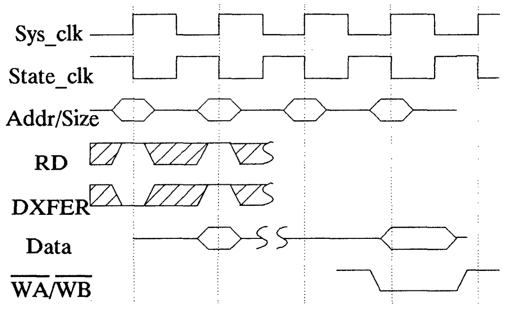


Figure 5. Actual Timing

CY7C330 Write-Enable Design

The signals required to generate the byte-write signals are shown in *Table 1*.

State_Clock

State_Clock is the inverted version of the System_Clock. It is used to drive the state registers in the CY7C330 PLD.

System_Clock

System_Clock is the clock that drives the IU and CY7C330 input registers. All transaction information is valid on the rising edge of this clock.

Table 1. Byte Write Signals

Name	Mnemonic
State_Clock	St_Ck
System_Clock	Sys_Ck
Advanced Write	WRT
Size(1:0)	Size1, Size0
Adr(1:0)	A1, A0
Adr14	A14
INULL	INULL
/Reset	!Rst
/Output Enable	!OE
/Write Enables - Bank A	!WA3 - !WA0
/Write Enables - Bank B	!WB3 - !WB0

Advanced Write

Advanced Write (WRT) is asserted (set to 1) by the processor during the first data cycle of single or double integer store instructions, and during the third cycle of atomic load/store instructions. WRT is send out "unlatched" and must be latched externally before it is used.

Size(1:0)

These two bits specify the data size associated with all transactions on the data bus. Size bits are sent out "unlatched" by the IU. The value of these pins corresponds to the data size corresponding to the memory address on the current cycle. The size bits are valid at the same time as the address bus. Since all instructions are 32-bits long, Size(1:0) is set to 10 during all instruction fetch cycles. Encoding of the Size bits is shown in *Table* 2.

Address (1:0), Address 14

The address bus is sent out "unlatched" and must be latched externally before use. If the Address Output Enable (/AOE) or Test Output Enable (/TOE) signals are de-asserted, the address bus tri-states. Address (1:0) is used to decode individual byte-write lines for writes within a 32-bit word boundary. Additionally, the CY7C330 design that follows, uses these lines to inhibit writes on unaligned boundaries. This feature could easily be modified to generate a memory exception, if so desired. Address 14 is used to select between Bank A write-enables (lower 16K words) and Bank B write-enables (upper 16K words) for the data RAMs.

Table 2. Size Bit Encoding

Size(1:0)	Transaction Type
00	Byte
01	Halfword
10	Word
11	Double Word

INULL

INULL has two meanings. First, it always occurs during the second cycle of a store transaction. When it occurs under these conditions, the signal is telling the memory subsystem that the current memory transaction has proceeded too far to be nullified, i.e., it is too late to initiate a wait-state or memory exception. Second, if the INULL occurs during the first cycle of a transaction, it is telling the memory subsystem to ignore the transaction entirely. This signal is of consequence only for store transactions that must be inhibited before the write occurs.

/Reset

This active low input to the CY7C330 PLD forces all outputs to the inactive state. It is a clocked reset.

/OE

This active low input to the CY7C330 PLD enables all outputs on the device. When high, all CY7C330 outputs are tristated. The ABEL source file containing the PLD equations for the CY7C330 Write-Enable Generator is shown in *Appendix A.*

CY7C332 Output Enable Design

The signals used to generate the required output enable signals are listed in Table 3. The design file for the CY7C332 PLD output enable circuit is provided in *Appendix B* and has been implemented using the Cypress PLD Toolkit, an assembler/simulator package developed by Cypress Semiconductor.

The design utilizes a CY7C332 to generate five instruction output enables and five data output enables for a Cypress SPARC-based non-cached memory system. Each output enable is decoded on a 16K word (word = 32-bits) boundary. The CY7C332 is especially well-suited for this application, since it incorporates input latch/registers with output decoding in a single PLD. When combined with a CY7C330 programmed as a write-enable generator, complete memory control is achieved in just two PLDs.

Table 3. Output Enable Signals

Name	Mnemonic
System_Clock	Sys_Ck
Size(1:0)	Size1, Size0
Adr(16:14)	A16,A15,A14
INULL	INULL
/Reset	!Rst
/Output Enable- > 332	!OE
/Output Enables - Inst Bank	!IOE4 - !IOE0
/Output Enables - Data Bank	!DOE4 - !DOE0
Inst Fetch Mem Exception	!IFMEMx

Pin 1 is the system clock, active on the rising edge. Pins 2-4 are address bits 16-14, which are used in the output enable decoding. Pins 5 and 6 are the IU size bits. For instruction fetches, if SIZE is not equal to '10' (0x2) then IFMEMx is made active. The SIZE bits are ignored for data fetches, since all alignment occurs in the IU. RD = 1 signifies that the following cycle is a read cycle. DXFER = 1 signals that the following cycle is a data transfer. Conversely, if DXFER = 0, the next cycle is a non-data (instruction) cycle. The INULL signal is not needed here, since the CPU ignores instruction/data fetched in the next cycle anyway. DOEx and IOEx are the data output enables and instruction output enables, respectively. IFMEMx occurs when an instruction fetch is attempted with SIZE not equal to '10' (1-word).

Conclusions

This application note shows that a non-cached memory subsystem for the Cypress CY7C601 SPARC IU can be easily implemented using the CY7C157, CY7C330, and CY7C332 devices. The design presented provides 128KB of instruction memory and 128KB of data memory with just ten components (eight CY7C157's, one CY7C330, and one CY7C332).

It is also important to realize that the memory could be easily expanded. The CY7C330 has four additional outputs, which could be used as write enables for an additional 64 KB of data memory. The CY7C332 design already provides output enables for 320 KB of data memory and 320 KB of instruction memory.

For systems requiring even larger memory spaces, a trade-off can be made with the CY7C330. If the smallest write boundary is changed to half-word (16-bits) instead of byte, the CY7C330 can provide byte writes for 384 KB of data memory. In a similar manner, for systems requiring only 32-bit writes to data memory, a single CY7C330 can provide the required write enables for 768 KB of memory! However, this would require an additional CY7C332 to decode output enables for data memory reads.

To summarize, for a system supporting byte writes to data memory, only ten components are needed to build a 128KB data and 128KB instruction memory subsystem. Using only one CY7C330 and one CY7C332, and adding only sixteen CY7C157s, a memory subsystem providing 320k bytes of instruction memory and 192KB of data memory can be constructed at a chip count of only eighteen devices! *Table 4* below tabulates the power characteristics of the memory subsystem.

Table 4. Memory Subsystem Characteristics

Component	Quantity	Power
7C157-33	8	1.375 W
7C330-50	1	0.99 W
7C332-20	1	0.99 W
TOTAL	10	13.0 W

Appendix A. ABEL CY7C330 Write Enable PLD Equations

```
Module SPARC_WRTENB flag '-r3'
title                                                          'SPARC Write Enable Generator'
LIBRARY 'P330';                                                "Enable various useful macros
IC device 'P330';

St_Ck,Sys_Ck,INULL, Rst                  Pin 1,2,10,13;        "Inputs
WRT,Size1,Size0,A1,A0,A14                Pin 3,4,5,6,7,9;

Reset, Set                               node 29, 30;          "Outputs and Internal Node declarations.
!WA3,!WA2,!WA1,!WA0,!WB3,!WB2,!WB1,!WB0  Pin 28,27,26,25,24,23,20,19;
!OE                                      Pin 14;
!WA3.OE istype 'Pin';                                          "Enable pin 14 as common OE for all outputs

SIZE = [Size1,Size0];          ADR = [A1, A0];           "Definitions for readability and test vector generation
WA = [ WA3,WA2,WA1,WA0 ];      WB = [ WB3,WB2,WB1,WB0 ];

H,L,C,X,Z = 1,0,.C.,.X.,.Z.;                             "Declarations

equations
WA3.OE = !OE;                                                  "Turn on outputs

WA3 := !Rst &!INULL &!A14 & WRT & (SIZE == 0) & (ADR == 3) #!Rst &!INULL &!A14 & WRT & (SIZE == 1) & (ADR == 2)
#!Rst &!INULL &!A14 & WRT & (SIZE == 2) & (ADR == 0) #!Rst &!INULL &!A14 & WRT & (SIZE == 3) & (ADR == 0)
#!Rst &!A14 & (SIZE == 3) & (ADR == 0)& WA3;

WA2 := !Rst &!INULL &!A14 & WRT & (SIZE == 0) & (ADR == 2) #!Rst &!INULL &!A14 & WRT & (SIZE == 1) & (ADR == 2)
#!Rst &!INULL &!A14 & WRT & (SIZE == 2) & (ADR == 0) #!Rst &!INULL &!A14 & WRT & (SIZE == 3) & (ADR == 0)
#!Rst &!A14 & (SIZE == 3) & (ADR == 0)& WA2;

WA1 := !Rst &!INULL &!A14 & WRT & (SIZE == 0) & (ADR == 1) #!Rst &!INULL &!A14 & WRT & (SIZE == 1) & (ADR == 0)
#!Rst &!INULL &!A14 & WRT & (SIZE == 2) & (ADR == 0) #!Rst &!INULL &!A14 & WRT & (SIZE == 3) & (ADR == 0)
#!Rst &!A14 & (SIZE == 3) & (ADR == 0)& WA1;

WA0 := !Rst &!INULL &!A14 & WRT & (SIZE == 0) & (ADR == 0) #!Rst &!INULL &!A14 & WRT & (SIZE == 1) & (ADR == 0)
#!Rst &!INULL &!A14 & WRT & (SIZE == 2) & (ADR == 0) #!Rst &!INULL &!A14 & WRT & (SIZE == 3) & (ADR == 0)
#!Rst &!A14 & (SIZE == 3) & (ADR == 0)& WA0;

WB3 := !Rst &!INULL & A14 & WRT & (SIZE == 0) & (ADR == 3) #!Rst &!INULL & A14 & WRT & (SIZE == 1) & (ADR == 2)
#!Rst &!INULL & A14 & WRT & (SIZE == 2) & (ADR == 0) #!Rst &!INULL & A14 & WRT & (SIZE == 3) & (ADR == 0)
#!Rst & A14 & (SIZE == 3) & (ADR == 0)& WB3;

WB2 := !Rst &!INULL & A14 & WRT & (SIZE == 0) & (ADR == 2) #!Rst &!INULL & A14 & WRT & (SIZE == 1) & (ADR == 2)
#!Rst &!INULL & A14 & WRT & (SIZE == 2) & (ADR == 0) #!Rst &!INULL & A14 & WRT & (SIZE == 3) & (ADR == 0)
#!Rst & A14 & (SIZE == 3) & (ADR == 0)& WB2;

WB1 := !Rst &!INULL & A14 & WRT & (SIZE == 0) & (ADR == 1) #!Rst &!INULL & A14 & WRT & (SIZE == 1) & (ADR == 0)
#!Rst &!INULL & A14 & WRT & (SIZE == 2) & (ADR == 0) #!Rst &!INULL & A14 & WRT & (SIZE == 3) & (ADR == 0)
#!Rst & A14 & (SIZE == 3) & (ADR == 0)& WB1;

WB0 := !Rst &!INULL & A14 & WRT & (SIZE == 0) & (ADR == 0) #!Rst &!INULL & A14 & WRT & (SIZE == 1) & (ADR == 0)
#!Rst &!INULL & A14 & WRT & (SIZE == 2) & (ADR == 0) #!Rst &!INULL & A14 & WRT & (SIZE == 3) & (ADR == 0)
#!Rst & A14 & (SIZE == 3) & (ADR == 0)& WB0;
```

Appendix A. ABEL CY7C330 Write Enable PLD Equations (continued)

```
"Test vectors for WA outputs, WB outputs are similar except for A14
"Note that the WA outputs are treated as active-high in the test vectors
"since they were declared as active-low in the pin declaration sections.

Test_vectors

([!OE,!Rst,St_Ck,Sys_Ck,WRT,INULL,SIZE,ADR,A14] -> [WA,WB]);      "!OE,!Rst,StCk,SyCk,W, I, S,ADR,A14 ]
[ 0,0, 0, 0, X, X, X, X, X ] -> [ X,X ];   [ 0,0, 0, 1, X, X, X, X, X ] -> [ X,X ];   [ 0,0, 1, 0, X, X, X, X, X ] -> [ 0,0 ]   ;"v1 Reset

"WRT = 0 = WAx inactive
[ 0,1, 0, 1, 0, 0, X, X, 0 ] -> [ 0,0 ];   [ 0,1, 1, 0, 0, 0, X, X, 0 ] -> [ 0,0 ];

"Halfword transactions to lower word (bytes 1:0)
[ 0,1, 0, 1, 1, 0, 1, 0, 0 ] -> [ 0,0 ];   [ 0,1, 1, 0, 1, 0, 1, 0, 0 ] -> [ 03,0 ];

"Halfword write on byte boundary results in IU generated alignment error.
[ 0,1, 0, 1, 1, 0, 1, 1, 0 ] -> [ 03,0 ];  [ 0,1, 1, 0, 1, 0, 1, 1, 0 ] -> [ 00,0 ];   "v10

"Halfword write to upper word
[ 0,1, 0, 1, 1, 0, 1, 1, 0 ] -> [ 03,0 ];  [ 0,1, 1, 0, 1, 0, 1, 1, 0 ] -> [ 00,0 ];   "v10

"Halfword write to upper word
[ 0,1, 0, 1, 1, 0, 1, 2, 0 ] -> [ 00,0 ];  [ 0,1, 1, 0, 1, 0, 1, 2, 0 ] -> [ 0c,0 ];
 [ 0,1, 0, 1, 0, 0, 1, X, 0 ] -> [ 0c,0 ]; [ 0,1, 1, 0, 0, 0, 1, X, 0 ] -> [ 00,0 ];

"Word write on byte bndary results in IU generated alignment "error
[ 0,1, 0, 1, 1, 0, 1, 3, 0 ] -> [ 0,0 ];   [ 0,1, 1, 0, 1, 0, 1, 3, 0 ] -> [ 0,0 ];

"Verify WA follows byte writes correctly [!OE,!Rst,StCk,SyCk,W,I,S,ADR,A14]
[ 0,1, 0, 1, 1, 0, 0, 3, 0 ] -> [ 0,0 ];                                              "v20
[ 0,1, 1, 0, 1, 0, 0, 3, 0 ] -> [ 08,0 ];  [ 0,1, 0, 1, 1, 0, 0, 2, 0 ] -> [ 08,0 ];  "wrt byte 3
[ 0,1, 1, 0, 0, 0, 0, 2, 0 ] -> [ 04,0 ];  [ 0,1, 0, 1, 1, 0, 0, 1, 0 ] -> [ 04,0 ];  "wrt byte 2
[ 0,1, 1, 0, 0, 0, 0, 1, 0 ] -> [ 02,0 ];  [ 0,1, 0, 1, 1, 0, 0, 0, 0 ] -> [ 02,0 ];  "wrt byte 1
[ 0,1, 1, 0, 0, 0, 0, 0, 0 ] -> [ 01,0 ];  [ 0,1, 0, 1, 0, 0, 0, 0, 0 ] -> [ 01,0 ];  "wrt byte 0
[ 0,1, 1, 0, 0, 0, 0, 0, 0 ] -> [ 00,0 ];                                             "writes are inactive

"Verify single store works correctly [!OE,!Rst,StCk,SyCk,W,I,S,ADR,A14] for ease of programming only

[ 0,1, 0, 1, 1, 0, 2, 0, 0 ] -> [ 0,0 ];   [ 0,1, 1, 0, 1, 0, 2, 0, 0 ] -> [ 0f,0 ];
[ 0,1, 0, 1, 0, 0, 0, X, 0 ] -> [ 0f,0 ];  [ 0,1, 1, 0, 0, 0, 0, X, 0 ] -> [ 0,0 ];          "v30

"Verify WA responds correctly to double stores
[ 0,1, 0, 1, 1, 0, 3, X, 0 ] -> [ 0 ,0 ];  [ 0,1, 1, 0, 1, 0, 3, X, 0 ] -> [ 0f,0 ];  [ 0,1, 0, 1, 0, 0, 3, X, 0 ] -> [ 0f,0 ];
[ 0,1, 1, 0, 0, 0, 3, X, 0 ] -> [ 0f,0 ];  [ 0,1, 0, 1, 0, 0, 2, X, 0 ] -> [ 0f,0 ];  [ 0,1, 1, 0, 0, 0, 2, X, 0 ] -> [ 0 ,0 ];

"Do the same thing for the WB outputs (no comments)
[ 0,0, 0, 0, X, 0, X, X, X ] -> [ X,X ];   [ 0,0, 0, 1, X, 0, X, X, X ] -> [ X,X ];   [ 0,0, 1, 0, X, 0, X, X, X ] -> [ 0,0 ];   "v1 Reset


"WRT = 0 = WA/B inactive
[ 0,1, 0, 1, 0, 0, X, X, 1 ] -> [ 0,0 ];   [ 0,1, 1, 0, 0, 0, X, X, 1 ] -> [ 0,0 ];

"Halfword transactions to lower word (bytes 1:0)
[ 0,1, 0, 1, 1, 0, 1, 0, 1 ] -> [ 0,0 ];   [ 0,1, 1, 0, 1, 0, 1, 0, 1 ] -> [ 0,03 ];
```

Appendix A. ABEL CY7C330 Write Enable PLD Equations (continued)

```
"Halfword write on byte boundary - occurence results in IU generated alignment error.
[ 0,1, 0, 1, 1, 0, 1, 1, 1 ] -> [ 0,03 ];                     "v10
[ 0,1, 1, 0, 1, 0, 1, 1, 1 ] -> [ 0,0 ];

"Halfword write to upper word
[ 0,1, 0, 1, 1, 0, 1, 2, 1 ] -> [ 0,0 ];       [ 0,1, 1, 0, 1, 0, 1, 2, 1 ] -> [ 0,0c ];
[ 0,1, 0, 1, 0, 0, 1, X, 1 ] -> [ 0,0c ];      [ 0,1, 1, 0, 0, 0, 1, X, 1 ] -> [ 0,0 ];

"Word write on byte boundary results in IU generated alignment "error
[ 0,1, 0, 1, 1, 0, 1, 3, 1 ] -> [ 0,0 ];       [ 0,1, 1, 0, 1, 0, 1, 3, 1 ] -> [ 0,0 ];

"Verify WB follows byte writes correctly
[!OE,!Rst,StCk,SyCk,W,I,S,ADR,A14]
[ 0,1, 0, 1, 1, 0, 0, 3, 1 ] -> [ 0,0 ]; "v20
[ 0,1, 1, 0, 1, 0, 0, 3, 1 ] -> [ 0,08 ];      [ 0,1, 0, 1, 1, 0, 0, 2, 1 ] -> [ 0,08 ];      "wrt byte 3
[ 0,1, 1, 0, 0, 0, 0, 2, 1 ] -> [ 0,04 ];      [ 0,1, 0, 1, 1, 0, 0, 1, 1 ] -> [ 0,04 ];      "wrt byte 2
[ 0,1, 1, 0, 0, 0, 0, 1, 1 ] -> [ 0,02 ];      [ 0,1, 0, 1, 1, 0, 0, 0, 1 ] -> [ 0,02 ];      "wrt byte 1
[ 0,1, 1, 0, 0, 0, 0, 0, 1 ] -> [ 0,01 ];      [ 0,1, 0, 1, 0, 0, 0, 0, 1 ] -> [ 0,01 ];      "wrt byte 0
[ 0,1, 1, 0, 0, 0, 0, 0, 1 ] -> [ 0,0 ];                                                      "writes are inactive

"Verify single store works correctly [!OE,!Rst,StCk,SyCk,W, I, S,ADR,A14] for ease of programming only
[ 0,1, 0, 1, 1, 0, 2, 0, 1 ] -> [ 0,0 ];       [ 0,1, 1, 0, 1, 0, 2, 0, 1 ] -> [ 0,0f ];
[ 0,1, 0, 1, 0, 0, 0, X, 1 ] -> [ 0,0f ];      [ 0,1, 1, 0, 0, 0, 0, X, 1 ] -> [ 0,0 ];

"Verify WB responds correctly to double stores
[ 0,1, 0, 1, 1, 0, 3, X, 1 ] -> [ 0 ,0 ];      [ 0,1, 1, 0, 1, 0, 3, X, 1 ] -> [ 0, 0f ];     [ 0,1, 0, 1, 0, 0, 3, X, 1 ] -> [ 0, 0f ];
[ 0,1, 1, 0, 0, 0, 3, X, 1 ] -> [ 0, 0f ];     [ 0,1, 0, 1, 0, 0, 2, X, 1 ] -> [ 0, 0f ];     [ 0,1, 1, 0, 0, 0, 2, X, 1 ] -> [ 0 ,0 ];

"Check that all WA's and WB's are inhibited when INULL occurs with WRT
[ 0,0, 0, 0, X, X, X, X, X ] -> [ X,X ];       [ 0,0, 0, 1, X, X, X, X, X ] -> [ X,X ];       "v1 Reset
[ 0,0, 1, 0, X, X, X, X, X ] -> [ 0,0 ];       [ 0,1, 0, 1, 1, 1, 0, 3, X ] -> [ 0,0 ];
[ 0,1, 1, 0, 1, 1, 0, 3, X ] -> [ 0,0 ];       [ 0,1, 0, 1, 1, 1, 0, 2, X ] -> [ 0,0 ]              ;"write byte 3
[ 0,1, 1, 0, 0, 1, 0, 2, X ] -> [ 0,0 ];       [ 0,1, 0, 1, 1, 1, 0, 1, X ] -> [ 0,0 ];             "write byte 2
[ 0,1, 1, 0, 0, 1, 0, 1, X ] -> [ 0,0 ];       [ 0,1, 0, 1, 1, 1, 0, 0, X ] -> [ 0,0 ];             "write byte 1
[ 0,1, 1, 0, 0, 1, 0, 0, X ] -> [ 0,0 ];       [ 0,1, 0, 1, 0, 1, 0, 0, X ] -> [ 0,0 ];             "write byte 0
[ 0,1, 1, 0, 0, 1, 0, 0, X ] -> [ 0,0 ];                                                      "writes are inactive

"Double stores
[ 0,1, 0, 1, 1, 1, 3, X, X ] -> [ 0 ,0 ];      [ 0,1, 1, 0, 1, 1, 3, X, X ] -> [ 0, 0 ]; "Inactive
[ 0,1, 0, 1, 0, 1, 3, X, X ] -> [ 0, 0 ];      [ 0,1, 1, 0, 0, 1, 3, X, X ] -> [ 0, 0 ];
[ 0,1, 0, 1, 0, 1, 2, X, X ] -> [ 0, 0 ];      [ 0,1, 1, 0, 0, 1, 2, X, X ] -> [ 0 ,0 ];

" MORE REALISTIC OCCURANCE OF DOUBLE STORE INULL
[ 0,1, 0, 1, 1, 0, 3, X, 0 ] -> [ 0 ,0 ];      [ 0,1, 1, 0, 1, 0, 3, X, 0 ] -> [ 0f, 0 ];     [ 0,1, 0, 1, 0, 1, 3, X, 0 ] -> [ 0f, 0 ];
[ 0,1, 1, 0, 0, 1, 3, X, 0 ] -> [ 0f, 0 ];     [ 0,1, 0, 1, 0, 0, 2, X, 0 ] -> [ 0f, 0 ];     [ 0,1, 1, 0, 0, 0, 2, X, 0 ] -> [ 0 ,0 ];

"Double stores
[ 0,1, 0, 1, 1, 0, 3, X, 1 ] -> [ 0 ,0 ];      [ 0,1, 1, 0, 1, 0, 3, X, 1 ] -> [ 0 ,0f ];     [ 0,1, 0, 1, 0, 1, 3, X, 1 ] -> [ 0 ,0f ];
[ 0,1, 1, 0, 0, 1, 3, X, 1 ] -> [ 0 ,0f ];     [ 0,1, 0, 1, 0, 0, 2, X, 1 ] -> [ 0 ,0f ];     [ 0,1, 1, 0, 0, 0, 2, X, 1 ] -> [ 0 ,0 ];

end SPARC_WRTENB
```

Appendix B. PLD ToolKit CY7C332 Output Enable PLD Equations

```
CY7C332;

CONFIGURE;

Sys_Ck,                                      {Pin 1 }
A16(ireg), A15(ireg), A14(ireg),             {Pins 2 thru 4}
SIZE1(ireg), SIZE0(ireg),                    {Pins 5 and 6}
RD(ireg), DXFER(node = 9,ireg),              {Pins 7 and 9}
                                             {Pin 8 is GND}
!OE(node = 14),                              {Pin 14 is out enb}
!DOE0(nenbpt),!DOE1(nenbpt),!DOE2(nenbpt),   {Pin 15 thru..}
!DOE3(nenbpt),!DOE4(nenbpt),                 {..19 }
!IFMEMx(node = 23,nenbpt),                   {Inst Fetch Mem Excp}
!IOE0(nenbpt),!IOE1(nenbpt),!IOE2(nenbpt),   {Pins 24 thru..}
!IOE3(nenbpt),!IOE4(nenbpt),                 {..28}

EQUATIONS;

IOE4 = RD & !DXFER & SIZE1 & !SIZE0 & !A16  & !A15 & !A14;     {A = 000}

IOE3 = RD & !DXFER & SIZE1 & !SIZE0 & !A16  & !A15 & A14;      {A = 001}

IOE2 = RD & !DXFER & SIZE1 & !SIZE0 & !A16  & A15 & !A14;      {A = 010}

IOE1 = RD & !DXFER & SIZE1 & !SIZE0 & !A16  & A15 & A14;       {A = 011}

IOE0 = RD & !DXFER & SIZE1 & !SIZE0 & A16  & !A15 & !A14;      {A = 100}

{ Recall that for Inst Fetches only SIZE(1:0) = '10' is allowed}
IFMEMx =    RD & !DXFER & !SIZE1 & !SIZE0                      {SZ = 00}
            RD & !DXFER & !SIZE1 & SIZE0                       {SZ = 01}
            RD & !DXFER & SIZE1 & SIZE0;                       {SZ = 11}

{DOE's do not depend on SIZE bits, since IU does alignment internally}
DOE4 =  RD & DXFER & !A16 & !A15 & !A14; {A = 000}

DOE3 = RD & DXFER & !A16 & !A15 & A14; {A = 001}

DOE2 = RD & DXFER & !A16 & A15 & !A14; {A = 010}

DOE1 = RD & DXFER & !A16 & A15 & A14; {A = 011}

DOE0 = RD & DXFER & A16 & !A15 & !A14; {A = 100}
                          {end of file}
```

NOTES:

CYPRESS SEMICONDUCTOR

Cache Memory Design

Introduction

The first commercial use of a cache memory was in 1969, the year IBM introduced the IBM 360/85. Since that time, cache memory has spread from mainframes to minicomputers to microcomputers, thus becoming an accepted design technique for a broad range of computing machines. Cache memory was conceptualized as an engineering solution to unacceptably high main memory access times relative to CPU cycle time, so high that main memory access time was severely limiting overall machine performance. This solution dictates logical placement of a small, high-speed buffer between the CPU and main memory. If this buffer (which is hidden from the outside world, thus the name *cache*) is designed properly, the machine will appear to have a large amount of very fast main memory. As an example of the effectiveness of this approach, consider a high-end machine like the Amdahl 580 or IBM 3090. This caliber of machine has a main memory access time of 200-500 ns and a cache access time of 20-50 ns, yielding an effective memory access time of 30-100 ns, a 5 to 7x increase in memory performance.

The use of cache memory has become very widespread as evidenced by cache being directly supported or included on-chip in a variety of microprocessors: the National Semiconductor 32000 family, the Motorola 68000 family, and the Intel 80386 and 80486, as well as all of the currently available RISC families such as the Cypress CY7C600 SPARC family. Clearly, an understanding of the functional attributes and engineering tradeoffs of cache design is in order.

The purpose of this application note is to serve that end in a general sense. The first section is a discussion of the cache design goal and methods of achieving that goal. Next, several main cache design factors are described, outlining the engineering tradeoffs or advantages/disadvantages of each factor. The cache concept is then extended to a multilevel hierarchy, including a discussion on the conditions for and techniques used in design of multilevel cache for uniprocessor and multiprocessor environments. The intent of this paper is to lay the groundwork for successive application notes that outline specific cache designs.

Cache Design

Cache Basics

The objective of cache design is to reduce the effective (or average) memory access time to some predetermined, acceptable level (generally determined from cost/performance tradeoff analysis). The mechanism through which this is accomplished can be identified by realizing that most processor reference streams are both highly sequential and highly loop-oriented. Therefore, a cache operates on the principle of spatial and temporal locality of reference. Spatial locality means that information that will be referenced by the CPU in the near future is likely to be logically close in main memory to information that is currently being referenced. Temporal locality means that information currently being referenced by the CPU is likely to be referenced again in the near future. Through these mechanisms, a cache is designed such that there is a high probability that CPU references will be located in the cache. Spatial locality of reference is serviced in the following manner: if the cache is referenced by the CPU, and does not contain the information requested (a "miss"), then the cache will access main memory and retrieve not only the information currently being requested, but also several additional locations that logically follow the current reference (this large set of information is called a "line" , or "block"). In this manner, the next CPU reference has a high statistical probability of being serviced by the cache, thus avoiding the relatively long access time of main memory. Temporal locality of reference is serviced by allowing information to remain in the cache for an extended period of time, only replacing the line in order to make room for a new one. There are several algorithms that can be used to manage cache line replacement, which will be discussed later. By allowing the information to remain in the cache and assuming sufficient cache size, it is possible for an entire loop of code to fit into the cache, thereby allowing very high speed execution of instructions in a loop.

The Cache Design Goal

The goal of a cache design is to reduce the effective memory access time as seen by the CPU. Effective access time can be expressed as:

$$t_{\text{eff}}=t_{\text{cache}}+m \times t_{\text{main}}$$

Where:

t_{cache}	=	effective "hit time" of cache C (ie, cache access time)
m	=	"miss rate" of cache C
t_{main}	=	Main memory access time (penalty beyond t_{cache} for main memory accesses)

Design of a cache revolves around:

- Minimizing the time for the cache to service a "hit"
- Maximizing the hit rate (obviously, hit rate = 1 - miss rate)
- Minimizing the delay due to a cache miss (included in t_{main})
- Minimizing the delay caused by overhead associated with keeping main memory coherent, especially in multicache configurations (included in t_{main})

Generally, all of the above factors are impacted in some way by each of the design parameters that will be discussed below. In an attempt to simplify the overall design process, it may be useful to view cache design from the following "macroarchitecture" viewpoints, each of which can be broken down into one or more "microarchitectural" parameters:

- Cache placement
  - physical vs. virtual cache
- Cache organization
  - cache mapping method
  - cache size
  - cache line size
  - split cache vs. combined cache
- Cache management
  - main memory coherence schemes
  - line replacement algorithms
  - fetching algorithms

The next several sections will examine, in the context of the macroarchitectural parameters, each of the microarchitectural aspects in detail, giving the performance trade-offs relative to the four cache performance factors identified above. After discussion of these design parameters, the critical parameters of cache design will be pulled together and a method of calculating ballpark estimates for effective cycle time will be presented.

Cache Placement

As stated earlier, the cache resides logically between the CPU and main memory. However, the cache is not the only functional block that resides in that location; an "address translation unit," usually called a Memory Management Unit (MMU), also sits between the CPU and main memory. The purpose of the MMU is to manage the mapping of virtual addresses (which are generated by a program and are used by the CPU) to physical addresses (which are used to access main memory). Cache placement refers to the location of the cache relative to the MMU. *Figure 1* shows two ways of

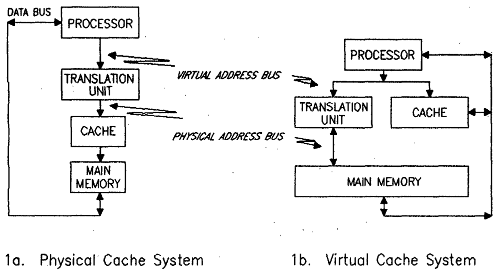


Figure 1. Cache Placement

arranging the cache and MMU. The issue boils down to "Where in the system should the MMU delay occur?" Traditionally, caches have been referenced with physical addresses, as in *Figure 1a.* The advantage of a physical cache is that it is easier to manage. The disadvantage is that it is slower than a virtual cache, which is referenced by virtual addresses as in *Figure 1b.* The reason that it is slower is that the address translation time is included in t_{cache}, which means that the translation delay occurs on every memory reference. A virtual cache system allows address translation to occur in parallel with cache access, thereby shifting the translation time penalty from t_{cache} to t_{main}, where the overall negative impact that it has on t_{eff} will be reduced significantly if the hit rate is high. Cypress's CY7C600 SPARC family utilizes a virtual caching scheme. The disadvantage of a virtual caching scheme is that the cache is more difficult to manage, since support must be included to detect and correct "aliases" (or synonyms). Aliasing occurs when two virtual addresses translate to the same physical address, and can occur, for example, when two different programs in the CPU share pages placed in different locations in the two programs' respective address maps. This "problem" can be detected and fixed in a number of ways. The most complete solution is to add dual cache tags (cache tags will be explained later) – a set of virtual cache tags and a set of physical cache tags – and use these two tables as a "cross-referencer" to detect and prevent aliasing. The CY7C605 CMU-MP uses this methodology. Another solution is to use an operating system detector that either forces shared data to the same cache line or marks shared data as non-cacheable. The CY7C604 CMU uses this technique. The bottom line is that aliasing is correctable, and as the demands placed on cache systems by faster processor speeds becomes more intense, virtual caching schemes will become more popular. Finally, consider the concept that as integration levels increase, more and more microprocessors will be available with on-board cache. In fact, several CISC (Complex Instruction Set Computer) chips already contain on-board cache (32000, 68030, 80486), and several RISC architectures have been proposed or introduced as a single chip with on-board cache. As a result, virtual cache vs. physical cache is likely to become a silicon design issue, with system-level designers focusing on methods of designing an efficient second-level cache to back up relatively small on-board cache. Second-level cache is defined hierarchically as a cache located between the cache accessed directly by the processor and main memory. In the event of a multilevel cache hierarchy, cache placement may or may not be an option. If the cache is on the processor chip, chances are that this will force a physical level 2 cache. There is also the probability that a multiple chip processor family will be partitioned in such a way that it forces a physical level 2 cache.

Cache Organization

Cache organization has four basic parameters: cache mapping method, cache size, cache line size, and split vs. combined cache. Note that for a multilevel cache hierarchy, the trade-offs associated with cache organization decisions regarding cache size, cache mapping method, and cache line size are multidimensional (and thus more complex) from the standpoint that choices made for the level 1 cache are likely to impact the performance of the level 2 cache – and vice versa.

Cache Mapping Method

Since a cache can be viewed as a (small) moving window into portions of a (larger) main memory, it is necessary to devise a scheme for mapping main memory locations to and from locations in the cache. The type of mapping that is used impacts both cache hit time and miss rate. Generally, an increase in hit rate exacts a penalty on cache hit time. However, recent research supports the idea that if a cache is sufficiently large, the relative difference in miss rate for various mapping methods becomes very small, indicating that a sufficiently large cache should be mapped according to the scheme that exacts the least penalty on cache hit time.

The most widely used mapping schemes are based on the principle of associativity. A fully associative cache allows any location in main memory to be mapped to any location in the cache. An *n*-way set associative cache (typically $n = 2, 4, 8$, etc) allows any particular location to be mapped to *n* locations in the cache. A direct-mapped cache allows any particular location in main memory to be mapped to only one location in the cache (i.e., it is a 1-way set associative cache). The following discussion details each technique, beginning with the least complex (direct-mapped) and finishing with the most complex (fully associative).

Direct Mapping

Figure 2 illustrates direct mapping. Each location in main memory maps to a unique location in the cache. For instance, location 1 in main memory maps to location 1 in the cache. Location 2 in main memory maps to location 2 in the cache. Location *m* in main memory maps to location *m* in the cache. Location *m + 1* in main memory maps to location 1 in the cache, etc. A simplis-

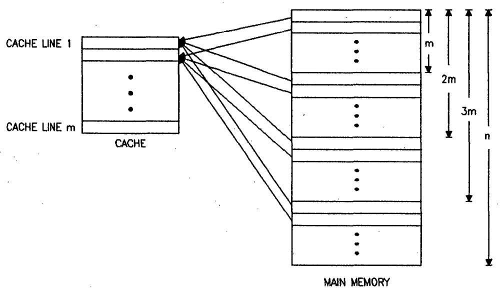


Figure 2. Direct Mapping

tic direct mapped cache implementation is shown in *Figure 3.* A direct-mapped cache consists of a data memory, a tag memory, and a comparator. The size of the data memory (which contains the cached data and instructions) is defined as the cache size. The tag memory is used to determine if the line being addressed by the processor is actually in the cache (via use of the comparator). The address is split into three fields: a tag field, an index field, and a word-offset field. The tag field consists of the higher-order bits of the address. The Index field is used to address the tag memory in order to see if the line being accessed is indeed the line that the processor desires. This mechanism ensures that, for example, data from (desired) cache location $2m+4$ is retrieved instead of data from cache location

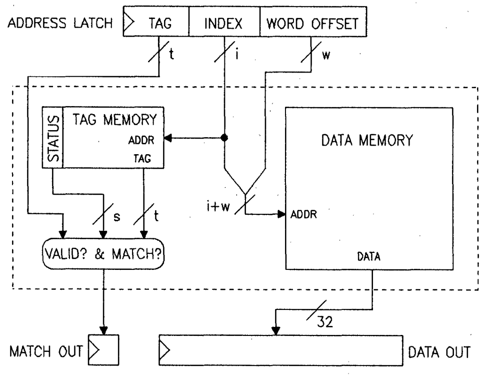


Figure 3. A Direct-Mapped Cache

$4m+4$ (which would reside in the same location in the cache). The line size is defined as the basic unit of transfer between the cache and main memory and is typically 16, 32, 64, 128, etc bytes. The number of bits in each field can be deciphered as follows:

$i = \log_2(\textit{\# cache tag entries})$

$w = \log_2(\textit{line size})$

$i + w = \log_2(\textit{cache size})$

$t = (\textit{\# address bits}) - i - w$

The cache functions in the following way: When an address is presented to the cache, the bits of the index field are used to address the tag store. The tag is accessed, and the tag contained in the location addressed by the index field is presented at its outputs. This tag is compared with the reference tag, while also checking to see that the status bits (i.e., VALID, DIRTY, etc.) are all right. In parallel with the tag access and status check, $i+w$ bits are used to address the data memory, with the accessed word being placed in the DATA OUT buffer. If the tags match and if the status bits check out all right, MATCH OUT is asserted, indicating that the information retrieved from the data memory is correct (a cache hit). If the tags do not match or if the status bits do not check out all right, MATCH OUT is deasserted (indicating that the data in DATA OUT is invalid and, therefore, a cache miss) and the correct data is retrieved from main memory. Consequently, a direct-mapped cache has two critical timing paths:

1. Read-data: accessing the data memory & passing the word to the DATA OUT register.
2. Asserting the MATCH OUT signal if the status bits are all right and the retrieved tag matches the reference tag.

Accordingly, cache access time for a direct-mapped cache is limited by the slower of paths 1 and 2.

Set Associative Cache Mapping

Figure 4 illustrates how set associative mapping works for the 2-way set associative case. The cache consists of 2 sets or banks of memory cells, each containing *m* lines. Location 1 in main memory maps to cache line 1 of either set. Location 2 in main memory maps to cache line 2 of either set. Location *m* in main memory maps to

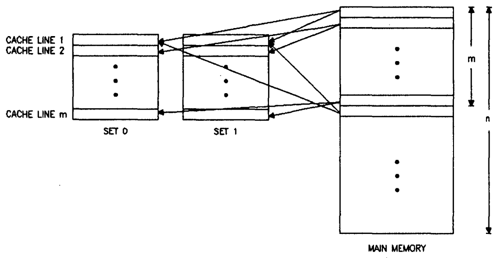


Figure 4. Set Associative Mapping

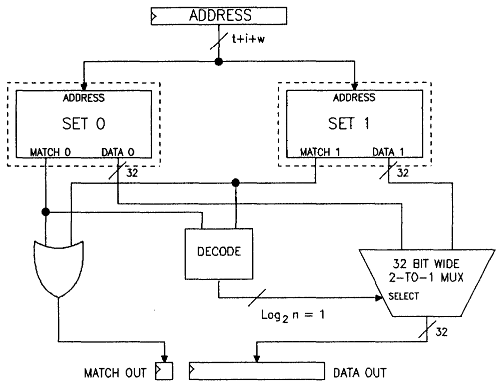


Figure 5. A 2-Way Set Associative Cache

cache line m of either set. Location $m+1$ in main memory maps to cache line 1 of either set. Location $m+2$ in main memory maps to cache line 2 of either set, and so on. In this manner, each location in main memory has 2 chances of being in the cache. This scheme allows, for example, main memory locations $m+z$ and $5m+z$ (where z is any integer) to coexist in the cache. This is an advantage in that it supports the principle of temporal locality of reference very efficiently for small cache sizes. This advantage goes away, however, when the cache becomes sufficiently large. *Figure 5* shows an implementation of an n-way set associative cache where $n=2$. Each of the sets, which are enclosed by a dashed block, contain the same logic that is inside the dashed block in *Figure 3* (direct-mapped cache). Additionally, an OR function is included to assert MATCH OUT if either set contains a match. The decode function selects data from the bank containing the match, and asserts a control line to the mux, thereby allowing the matched data to propagate to DATA OUT. Two comments: First, extension of this topology to n-way set associativity simply means having n sets of memory, and n-input OR function, an n-to-1 decoder, and an n-to-1 mux. Second, this is only one of several topologies. Another way of implementing the mux function would be to assert RAM output enables based on the outcome of the matching function. Yet another way would be to combine the OR and DECODE functions into one package (which could easily fit into a PLD). Obviously, there are more logic levels in a multi-way set associative cache than in a direct-mapped cache. A multiway set associative cache contains three critical timing paths:

1. Read-data: accessing the cache data memory in each of the sets.
2. Asserting the MATCH OUT signal in one of the sets if the tag is matched and valid.
3. Select-data: selecting the cached data from the set that matches (if there is a match).

Intuitively, multiway set associative caches are slower than a direct-mapped cache because of the added logic delay associated with the select-data path. Therefore, a direct-mapped cache will exhibit a faster cache hit time at a lower system cost.

Fully Associative Mapping

Figure 6 illustrates fully associative mapping. With a fully associative scheme, any location in main memory can be mapped to any location in the cache. This scheme theoretically produces the highest hit rate because there is no possibility of "thrashing." Thrashing occurs when two or more blocks of data that map to the same location in the cache start replacing each other frequently. The end result is a drastic increase in t_{eff} due to increased miss rate. Thrashing becomes statistically unlikely, however, as cache size increases. *Figure 7* illustrates a simplistic fully associative cache. As shown, the address accesses a CAM (Content Addressable Memory) bank that simultaneously searches all locations for a match. If a valid match is found, the cache data RAM places the requested information in DATA OUT. If a match is not found, main memory must be accessed for the correct data. Fully associative caches

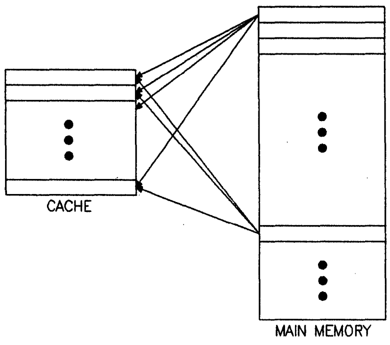


Figure 6. Fully Associative Mapping

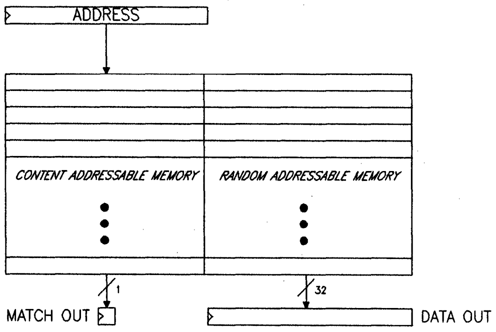


Figure 7. A Fully Associative Cache

are very expensive to build due to the fact that CAM cells are not readily available. Consequently, most caches are designed with direct or set-associative mapping, which can be realized with SRAM technology.

Design Trade-offs: Direct vs. Set Associative Mapping

The trend in cache design is toward larger caches. In the past, cache sizes of 8 KB to 16 KB were fairly common. Today, 64 KB is probably the average with many processors capable of supporting much larger cache sizes. As an example, consider the 80386. (a low-end processor) used in combination with the 82385 cache controller. The i82385 directly supports 32 KB cache size, and will indirectly support 64 KB and 128 KB cache. The '385 supports both direct mapped and 2-way set associative cache. When coupled with the Cypress CY7C184 Cache Data RAM (which was designed specifically for this application) a 32 KB cache can be realized with three chips – one 82385 and two CY7C184's. As another example, consider the Cypress CY7C600 SPARC family (a high-end processor family). This family supports direct-mapped cache in 64 kB "clusters," each consisting of one CY7C604 Cache Tag/Cache Controller/Memory Management Unit (CMU) and two CY7C157 16K x 16 Cache Data RAM's. Up to four clusters can be included per processor, effecting up to a 256 KB direct-mapped cache. Clearly, the industry trend is toward larger cache size.

There are two basic reasons for this: First, semiconductor technology is now capable of easily supporting 64 KB cache size with reasonable chip count and speed. Second, the emergence of multiple RISC (Reduced Instruction Set Computer) architectures demand higher cache hit rate and faster cache hit time; in other words a large, simply designed (i.e., fewer logic delays) cache. These trends, larger cache size and faster hit time, tend to favor easier-to-design direct mapped cache. The basic trade-off is that as associativity (which is defined as the number of cache lines in which a given block of data may reside) is reduced, fewer lines are searched on a memory reference. This provides a potential implementation advantage in that as fewer lines are searched, logic delay paths disappear and the cache gets faster. The downside to this is that as associativity decreases, the number of lines which have identical tags that can be simultaneously resident in the cache also decreases.

Valid arguments can be presented that support using set-associative mapping over direct mapping and vice versa. However, most researchers agree that the trend is toward direct mapping. There are two basic arguments against direct mapping: First, *direct-mapped cache has a lower hit rate than a set-associative cache of the same size*. This is a true statement, but is rapidly becoming a "don't care." Consider *Figure 8*.[1] For small cache size, direct mapping exhibits a considerably higher miss rate than either 2-way or 4-way set associative mapping. But for large cache size (64 KB) the miss ratio difference between direct mapping and set-associative mapping becomes a fraction of 1%. Research presented in [Hill]

1 Transcribed from the ACM Transactions on Computer Systems, 11/88, Vol. 6, No. 4, "Cache Performance of Operating Systems and Multiprogramming Workloads", (Agrawal, Hennesy, Horowitz)

shows that, for an 8 KB unified instruction/data cache, the difference in miss rate for 2-way set-associative vs. direct-mapped cache is around 1.3%. That figure drops to around 0.5% for 32 KB cache. The end result is that for large cache size, the reduced logic delay inherent in direct mapping (specifically, elimination of the Select-Data path) produces a cache that is faster and displays essentially the same hit rate as a similarly sized set associative cache. Thus, recent research supports the use of direct mapping.

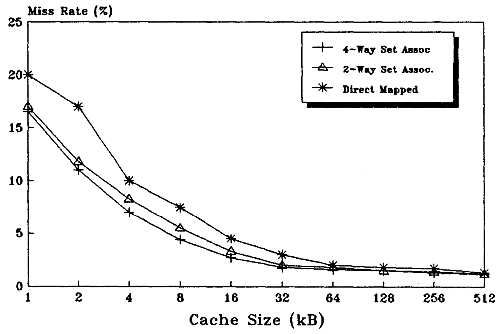


Figure 8. Cache Miss Rate as a Function of Associativity

The second argument against direct-mapped cache is that *a direct-mapped cache is more prone to "thrashing."* On the surface, this makes a good deal of sense. But for larger cache size, the statistical likelihood of this occurring is so low that it becomes negligible. Additionally, for real-time applications where deterministic response time (to a memory reference) is critical, the possibility of thrashing can be completely eliminated if cache entries can be "locked."

Four sound arguments can be presented in support of direct mapping: First, direct-mapped cache is less expensive than set-associative cache due to elimination of the logic associated with the Select-Data function. Second, the cache access time for a direct-mapped cache is faster than for a set associative cache due to elimination of logic delays associated with the Select-Data function. Third, t_{eff} is generally lower for a direct mapped cache than for a set associative cache for sufficiently large (generally 32 KB) cache size because t_{cache} is reduced and delta-m is negligible. Finally, there is no need for implementation of a cache line replacement policy for a direct-mapped cache since direct mapping is a one-to-one relationship (cache replacement policies will be discussed later).

Cache size

Cache size is perhaps the single largest influence on miss ratio, and also the most difficult to quantify in terms of miss ratio impact since the size of cache needed is so closely related to the principle of locality of reference and therefore the software workload. In general, however, a larger cache has a lower miss ratio. But large cache is also significantly more expensive to build given the relatively higher cost of fast SRAMs. In addition, mindlessly increasing the size of the cache can actually result in a performance drop. This performance drop may be the result of an increase in output loading due to fan-in/fan-out limitations or the increase in cache hit processing time due to added logic delays necessary to manage a larger cache. Given the current state of semiconductor technology, cache sizes of 64KB are easy to achieve, which is generally large enough to allow a cache to be designed with a 96% hit rate.

For multilevel cache hierarchies, a level 2 cache must, in general, be very much larger than the level 1 cache in order to be effective. Research results presented in [Short, Levy] indicate that addition of a second level of cache can provide a worthwhile performance increase given the proper combination of small-first-level cache and slow main memory.

Cache Line Size

Cache line size, which is defined as the basic unit of information transfer between the cache and main memory, ranks second, right behind cache size, as the parameter that most affects cache performance. Proper choice of line size is important because it impacts both miss rate and t_{main}. *Figure 9* presents data that has been transposed from [Smith]. Note that for a given cache size, increasing the line size reduces miss rate. But eventually miss rate begins to increase with larger line size (see the 2 KB curve in *Figure 9*). Cache line size also has an impact on t_{main}. Line sizes that are too large have long transfer times (thereby increasing t_{main}) and create difficulties in multiprocessing systems by generating excessive bus traffic. This is particularly true for primitive buses that do not support single-address, multiple-data cycle burst transfers. Newer bus protocols, such as Futurebus and Cypress's Mbus (Module bus), allow larger line sizes with less impact on t_{main} due to their burst transfer capabilities. Additionally, larger line sizes tend to effect a degree of "memory pollution." Memory pollution occurs when information is loaded into the cache, but is never referenced by the processor.

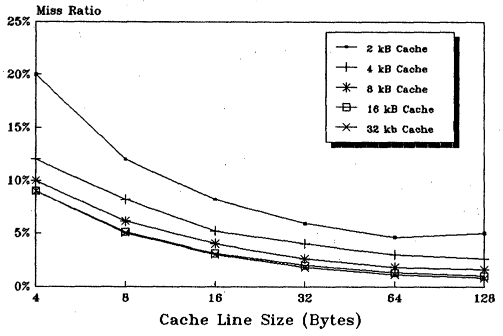


Figure 9. Cache Miss Rate as a Function of Line Size

For multicache organizations, having a level 2 cache line size that is greater than the level 1 cache line size has other advantages as well (that are not discussed in [Short, Levy]). Specifically, increased performance (due to the pre-fetch nature of the line-size difference) and lower cache tag cost. If the level 2 cache line size is greater than the level 1 cache line size, provision must be included to account for this. Generally, the ratio of level 2 cache line size to level 1 cache line size is set to be a power of 2. Recall that line size is defined as the basic size of information transfer between the cache and main memory (or between the level 1 cache and the level 2 cache). If the line size of the level 2 cache is not equal to the line size of the level 1 cache, the level 2 cache controller must be able to communicate in two different sizes of "data chunks." Using this type of sector-oriented cache, coherency is maintained in sizes equal to the level 1 cache line size (a "sub-block" of a level 2 cache line), meaning that the level 2 cache tag entries must include bits to track VALID, DIRTY, and INCLUSION (which indicates that the sub-block is present in the level 1 cache) for each sub-block. To illustrate this, consider a 16 KB direct-mapped level 1 cache with a 16 byte line size that is backed up by a 256 KB direct-mapped level 2 cache. If the level 2 cache line size is equal to the level 1 cache line size (e.g., 16 bytes), the level 2 cache will have (256K/16) or 16K cache tag entries. The tag size in bits (if a 32 bit address is assumed) is then 32 - $\log_2(16K)$ - $\log_2(16)$ or 14 bits plus 3 bits (for VALID, DIRTY, and INCLUSION) for a total of 17 bits long. This equates to a cache tag size of 16K x 17 or 272 Kbits tag size. If, on the other hand, the level 2 cache line size is set at 64 bytes, the level 2 cache will have 4K tag entries. The tag size would then be 14 bits plus the 3 status bits needed for each of the 4 sub-blocks in the level 2 cache line for a total of 26 bits of tag. The total tag size would then be 4K x 26 or 104 Kbits, meaning that the tag for the sector-based level 2 cache would cost 40% as much as the tag for the non-sector-based cache tag on a cost/bit basis. Therefore, in addition to the possible performance benefit associated with having a level 2 line size that is greater than the level 1 line size, the cache will be less expensive as well.

In summary, three factors influence cache line size choice:

1. The type of bus protocol that is used. Use of a protocol that is capable of burst transfers (such as Futurebus or Mbus) will permit a larger line size with a potential increase in performance.
2. The structure of main memory. In other words, make sure that the chosen line size will not create a bottleneck at the main memory interface.
3. Bus bandwidth/data contention considerations, especially in a multiprocessing environment.

The design task boils down to choosing a line size that is big enough to effect a good miss ratio, but small enough to minimize $t_{\mathbf{main}}$. Typically, cache line size is 16, 32, 64, or 128 bytes.

Split vs. Combined Cache

In the past, computers have generally utilized a single cache for both instructions and data. It is possible, however, to design a system that has separate caches for instructions and data. Generally, as shown in *Figure 10*,[2] a unified instruction/data cache results in slightly higher performance through a lower miss ratio. The advantages of splitting the cache are:

1. It makes design of the instruction cache easier since it's contents do not generally need to be modified.
2. It may eliminate conflict between data and instruction accesses in a pipelined architecture (this would depend, of course, on the overall processor architecture).

2 Transcribed from "CPU Cache Memories", 1984 (Smith, University of California, Berkely)

There are also advantages to using a unified cache:

1. Cache design is simpler for a unified cache because: (a) cache-to-main memory communications are one-to-one and (b)cache-to-processor communications are one-to-one.
2. A unified instruction/data cache tends to make more efficient use of the cache, which is a limited resource.

Cache Management

Cache management, in this context, refers to the policies that are used to move information into and out of the cache. These policies are not directly related to cache organization, but they do have an impact on the complexity of the cache controller. Specifically, cache management refers to the policies that:

1. Keep main memory coherent relative to cached information.
2. Determine when new information should be loaded into the cache from main memory.
3. Determine (if there is a choice available) which line in the cache should be replaced with the new information that is being loaded into the cache.

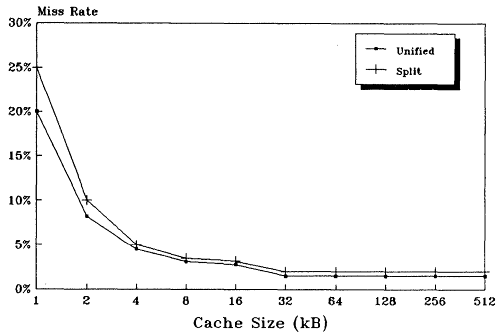


Figure 10. Miss Rate for Split Cache v.s. Combined Cache

Main Memory Coherence Schemes

When the CPU modifies data that is cached, main memory needs to be notified of the change at some point in time. Whether this happens "sooner" or "later" depends on the coherency scheme that is used. There are two mainstream coherency schemes: write-through and copy-back. Each policy has advantages and disadvantages, and each impacts both the complexity of the cache controller and t_{eff}. Using write-through, all writes to cached locations are immediately written through to main memory. This policy is the simpler of the two to implement, resulting in a less complex cache controller design. It can, however, result in a performance decrease since the CPU usually must be held pending completion of the write. Write-through can also cause problems due to increased bus traffic. The copy-back policy only updates the cache on CPU store cycles, updating main memory only when it becomes necessary to replace a modified (or "dirty") line in the cache. This policy requires an extra bit in the cache tag array to keep track of whether a line is "clean" or "dirty." The main advantage of copy-back is that it generates less memory bus traffic, resulting in higher performance. The main disadvantage of copy-back is increased complexity of the cache controller. *Table 1* outlines the major advantages/disadvantages of both policies. Additionally, a system can implement "write allocation". Write allocation means that on a write miss, the data addressed by the write miss is loaded into the cache and then modified. With no write allocate, the data is written to main memory only, and the cache is not updated.

For multilevel cache systems, reducing the overhead required to maintain consistency between the level 1 cache, the level 2 cache, and main memory is a critical design factor. The trade-off is one of cache controller complexity and the amount of bus bandwidth vs. cost. According to [Short, Levy], the level 1/level 2 cache coherency strategy could result in a 15% cache system performance differential. In a 2-level cache, choice of write strategy can generally be made independently of the level, the choice of strategies (from highest performance to lowest performance) being:

Level 1	Level 2
Copy-back	Copy-back
Copy-back	Write-through
Write-through	Copy-back
Write-through	Write-through

Table 1. Engineering Trade-offs: Write-through vs. Copy-back

+/–	*Write-through*	*Copy-back*
+	Main memory always has the most up-to-date version of data – minimizing cache coherency problems for multicache configurations.	Produces a lower miss rate than write-through for some applications
+	Easy to implement in the cache controller.	Frees up bandwidth on the main memory bus due to less frequent memory updates.
–	Without buffering (eg, posted writes), CPU must wait for write to complete.	Difficult to realize in multiprocessing systems due to cache coherency issues.
–	If write buffers are present, extra logic must be included to ensure that data will not be referenced from main memory until it has been stored there.	Extra logic needed for DIRTY bit.
–	Generates increased bus traffic, which is particularly bad for multiprocessing systems.	Results in a more complex controller design, since it caches writes in addition to reads.

Line Replacement Algorithms

The function of the line replacement algorithm is to decide which entry in the cache will be replaced when a new line must be loaded into the cache. For a direct mapped cache this task is very straightforward, since each main memory location maps to a unique line in the cache. For set-associative cache (fully associative will not be discussed), there is some latitude as to which set will have a line replaced. The most common methods of replacing cache lines are Least Recently Used (LRU) and First In/First Out (FIFO). The LRU algorithm keeps track of which set contains the line that has gone the longest without being used, and replaces that line. The FIFO algorithm keeps track of which set contains the oldest line, and replaces that line. It is also possible to use a random cache line replacement algorithm, where the set containing the line to be replaced is chosen at random. Curiously, research presented in [Smith, Goodman] shows that random replacement generally produces higher hit ratios than either the LRU or FIFO algorithms. *Figure 11* was created using data from [Smith, Goodman] and shows relative hit ratios for 4-way set associative LRU and random, 2-way set associative LRU and random, and direct replacement (for direct-mapped cache). Two notes of caution: First, this data is fairly old (1983) and therefore does not show data for reasonable cache size (by today's standards). Second, this data was obtained by averaging trace data from three different C programs running under UNIX on a VAX-11/780, so depending on this data absolutely would be inappropriate (especially for RISC machines). Rather, relative comparison of each policy and cache organization is most appropriate.

Some interesting conclusions can be drawn from the data that is presented in *Figure 11.* First, the random replacement algorithm appears to provide nearly the same or better hit rates than LRU. This is significant because a random replacement algorithm is very much easier to design into a cache controller and requires less hardware. The second conclusion is that for 8 KB (and presumably larger) cache size, direct mapped cache offers nearly the same hit ratio performance as 2-way and 4-way set associative cache. This supports the conclusions drawn in the section on cache mapping techniques.

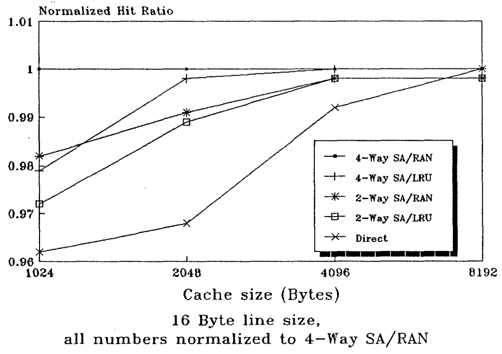


Figure 11. Cache Hit Rate as a Function of Replacement Algorithm

Fetching Algorithms

Most caches use demand fetching, where a new line is requested from main memory only when a CPU reference results in a cache miss. This method results in a less complex cache controller design. An alternate method, which can produce higher hit rates in some applications, is pre-fetching. Pre-fetching makes use of idle memory cycles to move data into the cache. Static pre-fetch is implemented at compile time, while dynamic pre-fetch occurs at run time. Sequential dynamic pre-fetching can cut miss rate in half according to [Smith]. [Kabakibo, et all] estimate a reduction in miss rate of as much as 75%-80%. This points to a significant performance advantage, but it requires a large cache size to be effective. The reason for this is that dynamic pre-fetch can result in increased memory pollution, and the statistical likelihood of this happening increases dramatically for decreasing cache size. Therefore, if cache size is large and cache controller complexity is not a major issue, inclusion of a dynamic pre-fetch mechanism can result in a significant performance increase. In a multilevel cache hierarchy, one way to increase the hit rate of the level 2 cache would be to implement a pre-fetch mechanism. Since the level 2 cache hit rate is usually fairly low anyway (generally 50% to 90%), memory pollution introduced by pre-fetch tends to be a don't care. This pre-fetch could be implemented with minimal hardware overhead by making the line size of level 2 greater than the line size of level 1.

Pulling it all together

This section will provide a simplistic method of calculating t_{eff} and the performance improvement of cache vs. no cache given various assumptions and design choices. Note that this methodology only provides "ballpark" figures. More accurate figures could be obtained by simulating an actual design either directly or via a software model.

As presented earlier, the goal of cache design is to reduce the effective memory access time (t_{eff}) as seen by the CPU. Effective access time is defined as

$$t_{eff} = t_{cache} + m \times t_{main}$$

The following methodology does not take into account the effects of design choices on t_{cache} or t_{main} – ie, these numbers are either already known or will be estimated. This methodology does, however, take into account the following factors via their effect on miss rate:

- Cache size
- Cache line size
- Cache mapping scheme
- Main memory coherency algorithm

This is accomplished by modeling the miss rate as

$$m = M \times MRM + CF$$

where:

m	=	Cache miss rate
M	=	"Raw" miss rate
MRM	=	Miss Rate Multiplier
CF	=	Coherency Factor

The raw miss rate is miss rate strictly as a function of cache size and cache line size, and is looked up in *Table 2*[3] (which assumes direct mapped cache). The Miss Rate Multiplier is essentially a "fudge factor" that accounts for variations in miss rate between direct mapped and set associative cache organizations, and is looked up in *Table 3*. The Coherency Factor is included to account for variations in miss rate due to the choice of main memory coherency algorithm. Recall that the write-through policy does not cache CPU writes. Write-through forces all CPU writes to immediately pass through to main memory. Thus, CPU writes to a write-through cache can be regarded as cache misses, meaning that $CF > 0$. If the cache uses write-through with posted write capability or uses the copy-back algorithm, CPU writes can be considered as cache hits, meaning $CF = 0$. CF is obtained by determining (or assuming) the percentage of cache references that are writes, and then derating the miss rate by that factor.

As an example, consider a single cycle 64 KB direct mapped cache with 32 byte line size that uses write-through and with 30% of cache references being writes.

3 Derived from "A Case for Direct Mapped Caches," IEEE Computer, 1988 (Hill) and "Line (Block) Size Choice for CPU Memories," IEEE Transactions on Computers, 1987 (Smith).

Table 2. Miss Rate as a Function of Cache Size and Cache Line Size

Cache Size (kB)	*Cache Line Size (Bytes)*					
	8	16	32	64	128	256
2	0.154	0.116	0.092	0.080	0.084	0.088
4	0.116	0.086	0.074	0.064	0.061	0.065
8	0.096	0.073	0.060	0.053	0.050	0.045
16	0.086	0.064	0.054	0.047	0.044	0.039
32	0.081	0.060	0.051	0.044	0.041	0.036
64	0.079	0.057	0.050	0.043	0.040	0.035
128	0.077	0.056	0.049	0.042	0.039	0.034
256	0.076	0.055	0.048	0.041	0.038	0.033
512	0.075	0.054	0.047	0.040	0.037	0.032

Table 3. Cache Miss Rate as a Function of Cache Size and Mapping Method

Mapping Method	*Cache Size (kB)*								
	2	4	8	16	32	64	128	256	512
Direct Mapped	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
2-Way Set Assoc.	0.975	0.980	0.986	0.990	0.994	0.995	0.996	0.996	0.996
4-Way Set Assoc.	0.925	0.940	0.958	0.970	0.982	0.985	0.988	0.989	0.989

Assume a 15-cycle main memory access time. From *Table 2*, $M = 0.050$. From *Table 3*, $MRM = 1.000$. $CF = 0.300$ (given). Then

$$\begin{aligned} m &= M \times MRM + CF \\ &= (0.050)\,(1.000) + 0.300 \\ &= 0.350 \end{aligned}$$

and

$$\begin{aligned} t_{\text{eff}} &= t_{\text{cache}} + m \times t_{\text{main}} \\ &= 1 + (0.350)\,(15) \\ &= 6.25\ \textit{cycles} \end{aligned}$$

meaning that this system will achieve a 2.4x performance increase if the cache as described is used. Note that the same system with a copy-back cache would achieve a t_{eff} of 1.75 cycles, meaning an 8.57x performance improvement. Finally, consider a 2-way set associative cache using copy-back. Now $t_{\text{eff}} = 1.746$ cycles and a performance improvement of 8.59 (which is less than 0.2% better than a direct mapped cache).

Multilevel Cache

Recent advances in silicon technology have forced a new focus in cache design methodology. Increased gate densities in integrated circuits have helped create the situation where it is now possible to include a small-to-medium-sized cache on the CPU chip itself. Examples of this include the Motorola 68030/040, the Intel 80486, and Intel's i860. Also, recent advances in silicon technology have permitted the introduction of ICs that support multiprocessing in a straightforward manner, meaning that multiprocessing systems will become more common. Examples of this include the Intel 80486 and the Cypress CY7C600 RISC family. Both of these issues tend to support a multilevel cache hierarchy. There are basically four factors that support a move to a multilevel cache hierarchy:

1. The actual implementation of the on-chip cache can force a cache partition. Specifically, a small or insufficiently large on-chip cache (with unacceptable or marginally acceptable hit rates) may force the addition of a second level of cache off-chip to achieve the performance objectives of a design. However, if the on-chip cache is designed improperly, a multilevel cache may be impossible or impractical. The on-chip cache must have the necessary "hooks" to permit communication between the first-level and second-level caches. If these "hooks" are not present, the user will be forced to accept lower performance in return for higher integration. An example of this is the Intel i860.

2. Detailed study of t_{eff} supports the statement that a multilevel cache hierarchy can offer higher performance than a single level cache hierarchy, particularly if the difference between processor speed and memory speed is very high. This speed difference may not be created solely from increases in CPU speed, but could also be the result of larger (and therefore slower) main memory.

3. Creating multiple cache levels also creates the possibility of functionally tuning each cache level for highest performance. For example, the first level cache could be optimized to minimize t_{eff}, and the second level cache could be optimized for high hit ratio, reduced cost, or reduced interconnect traffic.

4. Increased usage of multiprocessing may force a multilevel cache hierarchy. Generally, each processor needs it's own cache (especially if it is a RISC engine) to increase performance and decrease bus traffic (bus bandwidth being an especially valuable resource in multiprocessing systems). Addition of a second level cache can be used to further reduce t_{eff}, particularly if the level 1 cache does not meet performance objectives.

The issue that must be resolved is the cost vs. performance tradeoff of multilevel vs. single level cache. This tradeoff is a function of the processor architecture, the on-board cache (if there is one), the structure of main memory, and the type of connection between the cache and main memory. Consequently, there are no set rules to justify inclusion of a multilevel cache hierarchy. However, recall that cache memory was created in the '60s as an engineering solution to performance problems stemming from extremely fast CPU speeds relative to main memory access times. [Kabakibo, et al] state that this gap needs to be a "factor of 10" before inclusion of a cache is justified and extend this to a "factor of 40" before a multilevel cache is justified. The actual ratio that justifies inclusion of a multilevel cache hierarchy is a personal decision (generally as much a marketing decision as an engineering decision) and making concrete statements regarding justification is not valid.

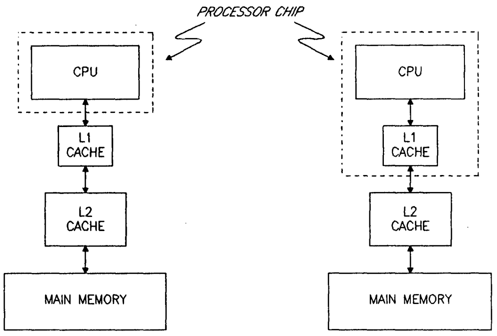


Figure 12. Multilevel Cache Hierarchy for Single Processor Systems

The balance of this paper will focus on multilevel cache hierarchies for uniprocessor and multiprocessor systems. In both cases, however, the hierarchy will be limited to two levels.

Multilevel Cache in Single-Processor Systems

The cache hierarchy that will be discussed in this section is presented in *Figure 12*. In this scheme, the level 1 cache services processor references and obtains data on a miss from the level 2 cache. The level 2 cache services references from the level 1 cache and obtains data on a cache miss from main memory. The level 1 cache can be inside or outside the processor chip. The design goal and methods of achieving that goal have not changed, there are simply more variables in the equations. The effective memory access time can be expressed as:

$$t_{eff} = t_{L1} + m_{L1}\,(t_{L2} + m_{L2} \times t_{main})$$

where:

t_{L1}	=	Level 1 cache access time.
t_{L2}	=	Level 2 cache access time (penalty beyond t_{L1}).

m_{L1}	=	Level 1 cache miss rate.
m_{L2}	=	Level 2 cache miss rate.
t_{main}	=	Main memory access time (penalty beyond t_{L2}).

Minimizing the delay caused by overhead associated with maintaining cache consistency is much more complex for multilevel cache hierarchies than for a single level cache hierarchy. When beginning a multilevel cache design, all of the previously discussed design factors must be carefully considered, but all of these design factors are now multidimensional problems.

The miss rate approximation presented earlier can be extended for two levels of cache. As an example, consider a system with a single cycle 2-way set associative level 1 cache that is 8 KB with a 32 byte line size and uses copy-back and a single cycle direct mapped level 2 cache that is 128 KB with a 128 byte line size and uses write-through with 30% writes. Main memory access requires 20 cycles. The effective memory access time for both level 1 and level 2 cache will be compared with access time for level 1 cache alone.

First, m_{L1} can be calculated from data in *Tables 2* and *3* to be:

$$\begin{aligned} m_{L1} &= M_{L1} \times MRM_{L1} + CF_{L1} \\ &= (0.060)(0.986) + 0 \\ &= 0.059 \end{aligned}$$

Then

$$\begin{aligned} t_{eff}\,|_{L1\text{ only}} &= t_{L1} + m_{L1} \times t_{main} \\ &= 1 + (0.059)(20) \\ &= 2.18\ \textit{cycles} \end{aligned}$$

Next, m_{L2} can be calculated as:

$$\begin{aligned} m_{L2} &= M_{L2} \times MRM_{L2} + CF_{L2} \\ &= (0.039)(1.000) + 0.300 \\ &= 0.339 \end{aligned}$$

Then

$$\begin{aligned} t_{eff}\,|_{L1\,\&\,L2} &= t_{L1} + m_{L1}\,(t_{L2} + m_{L2} \times t_{main}) \\ &= 1 + 0.059\,[1 + 0.339\,(20)] \\ &= 1.46\ \textit{cycles} \end{aligned}$$

So, the percent performance improvement over using only the level 1 cache is 33%. Also, note that our model produces a t_{eff} of 1.459 cycles for a 2-way set associative level 2 cache, which results in trading a more complex, more expensive cache controller design for essentially no performance improvement over a direct mapped implementation. Also, if the level 2 cache is direct-mapped and uses copy-back, t_{eff} is 1.11 cycles, resulting in nearly a 50% improvement over using only the level 1 cache.

Multilevel Cache in Multiprocessing Systems

Multiprocessing systems are becoming more and more prevalent in the industry. The obvious reason for this is to allow the rate of growth of computer system technology to be higher than the rate of growth of processor technology. The single most performance-limiting factor in multiprocessing systems is maintaining consistency between a global main memory (a global main memory being desireable to programmers) and multiple processors each having its own cache. Adding a second level of cache may aggravate this consistency problem, and in fact may cause a degradation in performance. However, a multilevel cache hierarchy can increase performance if implemented properly.

Multicache Consistency in Multiprocessing Systems

In multiprocessing systems, it is generally preferable for each processor to have a private cache to minimize bus traffic and a common global main memory to support ease of programming. Since a cache system works by providing a (small) local window into a (larger) main memory, and since a multiprocessing environment generally consists of more than one of these local windows, there is a possibility (in fact a definite probability) that more than one cache can contain the same data. Furthermore, if more than one cache shares a piece of data, and one of them should happen to change that piece of data, one or more of the caches will contain incoherent data. Therefore, a set of rules (e.g., a multicache consistency protocol) must be created to manage the task of maintaining consistency. As described earlier, maintaining coherence in a uniprocessor system with a single level of cache is fairly simple because coherence only needs to be maintained between one cache and main memory, and can be realized by implementing the copy-back or the write-through protocol. The consistency problem is more complex in multiprocessing systems where each processor has a private cache because consistency must be maintained between a cache, its "sibling" caches, and main memory. The consistency problem in this case (while more complex) is well defined and has well-known solutions. Typically, for a multiprocessing system with a large number of processing elements, a software consistency protocol is implemented. For systems with a small-to-medium number of processing elements, a bus-based protocol is usually implemented. Addition of a second level of cache tends to aggravate the consistency problem by introducing another level at which consistency must be maintained. Since multicache, multiprocessor topologies have some combination of multiple level 1 caches interfacing to a single level 2 cache and/or multiple level 2 caches interfacing to a common global main memory, there must be a component in the effective memory access time equation to account for time wasted while attempting to gain access to a "parent memory." Therefore, the effective memory access time equation for multiprocessing systems with multilevel cache hierarchies contain an additional term to account for "consistency management traffic." The position at which this time delay enters the equation depends on the topology used. Minimization of this contention delay as well as minimization of the delays caused by consistency management is critical to cache design in multiprocessing systems with a multilevel cache hierarchy. The rest of this section will consist of a discussion of how this extra level of coherence management impacts system performance.

Three different multiprocessing topologies are presented in *Figure 13*. Most authors agree that the level 2 caches should be supersets of their children caches. In this manner, the coherence management protocol can be moved as far away from the processing element as possible, thus allowing the level 2 caches to shield the level 1 caches from unnecessary blind checks and invalidations that may propagate up from main memory. The Multilevel Inclusion (MLI) Principle is stated for set associative caches in [Baer, Wang]. As stated, MLI can be achieved if the degree of set associativity of a parent (level 2) cache is greater than or equal to the product of the number of its children (level 1) caches, their degree of set associativity, and the ratio of their block sizes. Expressed mathematically:

$$Set\,Associativity_{L2} = \sum^{\text{All L1s}} \left[\, Set\,Associativity_{L1} \times \frac{Line\,Size_{L2}}{Line\,Size_{L1}} \right]$$

Note that MLI is not a requirement in multicache designs, and furthermore, the scheme proposed in [Baer, Wang] is only one of several ways to achieve MLI. As will be shown, MLI as stated in [Baer, Wang] is very restrictive and results in an extremely complex and expensive level 2 cache design. As an example of this, consider that to enforce MLI according to the

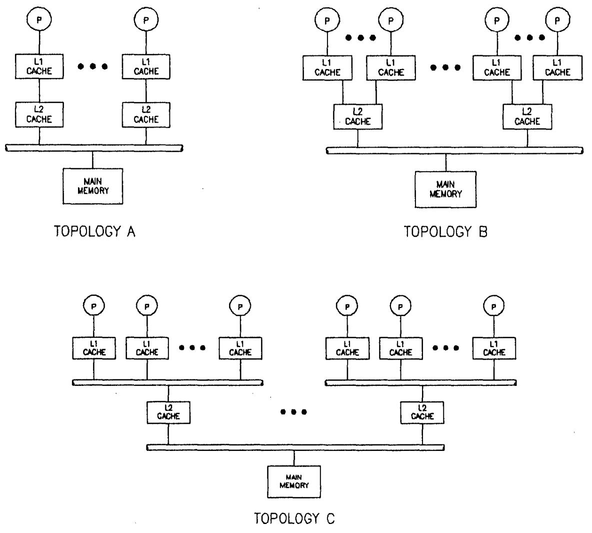


Figure 13. Multiprocessing Topologies with Multilevel Cache Hierarchies

scheme presented in [Baer, Wang], the example given in the section on Multilevel Cache in Single Processor systems would dictate an 8-way set associative level 2 cache. This may be an unrealistic goal from a cost standpoint, since an 8-way set-associative 128 KB cache would be very expensive to implement. However, for Topology A in *Figure 13*, MLI can be effected under the scheme presented in [Baer,Wang] if, for example, the level 1 cache is a direct mapped 16 KB cache with 16 byte line size, and if the level 2 cache is a 4-way set-associative 256 KB sector-based cache with 64 byte line size. Additionally, using Topology A a simple cache coherence protocol such as copy-back or write-through can be implemented at the level 1 cache (which is generally small). Cost effectiveness may dictate a fairly large sector-based level 2 cache. Consistency among the level 2 caches is maintained on a level 1 cache line-size basis. All level 1 cache misses are serviced by a private level 2 cache. Only when a sub-block in a level 2 cache (that has its INCLUSION bit set) needs to be replaced does the level 1 cache need to be disturbed. Performance can be increased dramatically if the bus can support direct data intervention (more on this later) and if the level 2 cache controller has a bus snooping mechanism that allows it to monitor bus activity and perform invalidations based on observed bus traffic. The effective memory access time for this topology is:

$$t_{\text{eff}} = t_{\text{L1}} + m_{\text{L1}}\,[t_{\text{L2}} + m_{\text{L2}}\,(t_{\text{main}} + t_{\text{bus, L2–main}})]$$

where $t_{\text{bus, L2-main}}$ is defined as the time required for a given level 2 cache to acquire the bus. The advantage of this topology is that it is simple and fairly straightforward to implement. The main disadvantage of this topology is that the level 2 cache is not shared by several level 1 caches.

Topology B, which depicts a multiport level 2 cache connected to multiple other level 2 caches via a bus, is probably the least desireable of the three topologies shown for several reasons. First, note that this topology contains two points at which contention may be experienced, resulting in an effective memory access time equation of:

$$t_{\text{eff}} = t_{\text{L1}} + m_{\text{L1}}\,[(t_{\text{L2}} + t_{\text{contention, L1–L2}}) \\ + m_{\text{L2}}\,(t_{\text{main}} + t_{\text{bus, L2–main}})]$$

where $t_{\text{contention, L1-L2}}$ denotes the arbitration/contention penalty for a level 1 cache to be serviced by a level 2 cache. Thus, this cache will be slower than topology A. Additionally, the logic required for arbitration at level 2 among the several level 1 caches will be expensive. Finally, note that MLI is very difficult to obtain for this type of system. Consider a system with four 16 KB direct mapped level 1 caches that have a 16 byte line size connected to a 256 KB level 2 cache that has a 64 byte line size. Given these parameters, the scheme proposed by [Baer, Wang] would dictate that the level 2 cache be 16-way set associative.

Topology 3, which is a bus-based hierarchy, is probably the most attractive topology for systems with a small-to-medium number of processing elements. Using this scheme, MLI is guaranteed through the use of *broadcast invalidations*. The effective memory access time for this topology is given by:

$$t_{\text{eff}} = t_{\text{L1}} + m_{\text{L1}}\,[(t_{\text{L2}} + t_{\text{bus, L1–L2}}) \\ + m_{\text{L2}}\,(t_{\text{main}} + t_{\text{bus, L2–main}})]$$

where $t_{\text{bus, L1-L2}}$ is defined as the time required for a given level 1 cache to acquire the bus between the level 1 caches and the level 2 cache. If the buses shown are architected properly (like Futurenet or MBus), bus traffic can be reduced to a minimum. The disadvantages of this topology are that it introduces greater hardware complexity, and that it really needs VLSI chips to be manageable (VLSI solutions such as this are available in the Cypress CY7C600 family). With a good bus protocol, the amount of bus traffic will still limit the number of resources that can share the bus. Even with these disadvantages, a bus-based multilevel cache hierarchy appears to be the most promising in terms of cost and performance considerations.

Multilevel Cache in SPARC Multiprocessing System

The Cypress CY7C600 RISC microprocessor family contains full support for multiprocessing, including an excellent bus-based multicache consistency mechanism. This section will cover the CY7C600 family members that comprise an “MP Cluster”; specifically, the CY7C601 Integer Unit (IU), the CY7C602 Floating Point Unit (FPU), the CY7C605 Cache Tag-Cache Controller-Memory Management Unit for Multiprocessing (CMU-MP), and the CY7C157 16K x 16 Cache RAM. In particular, the features of the CY7C605 CMU-MP that support multicache consistency will be highlighted. Additionally, a section is included on Mbus. Finally, a SPARC multiprocessing system will be extended to a multilevel cache hierarchy (demonstrated in two topologies). These topologies will then be examined, with a focus placed on implementation and performance advantages/disadvantages.

The SPARC Multiprocessing Cluster
As presented in *Figure 14*, the basic SPARC Multiprocessing (MP) cluster consists of a CY7C601 IU, a + CY7C602 FPU, a CY7C605 CMU-MP, and two CY7C157 Cache RAMs. The cache size can be increased by adding up to three more CY7C605s and 6 more CY7C157s as shown in *Figure 15*, thereby allowing cache sizes from 64 KB to 256 KB in 64 KB increments. Also, several MP clusters can be connected over the M-bus (Module-bus) to form a multiprocessing system as shown in *Figure 16*.

The CY7C601 Integer Unit. The CY7C601 IU is fully compliant with the SPARC reference Instruction Set Architecture, contains full support for eight register windows, full IEEE floating point co-processor interface in addition to a second generic (user-defined) coprocessor interface. The device is available at 25, 33, and 40 MHz and is implemented in a 0.8 micron dual layer metal CMOS process.

The CY7C602 Floating Point Unit. The CY7C602 FPU is a single chip SPARC floating point processor with full IEEE double precision support, a dedicated register file, 64 bit data paths, and is available at up to 40 MHZ.

The CY7C157 Cache Data RAM. The CY7C157 Cache RAM is a custom design for CY7C604 and CY7C605 cache systems (but is still a fairly generic cache RAM). It is a fully synchronous (eg, self-timed) device that is organized as 16kx16, which is much better suited to cache design than "industry standard" asynchronous RAM's. The '157 is designed such that it will scale in speed, matching the clock rate of the IU and CMU. It is also implemented in 0.8 micron dual layer metal CMOS technology.

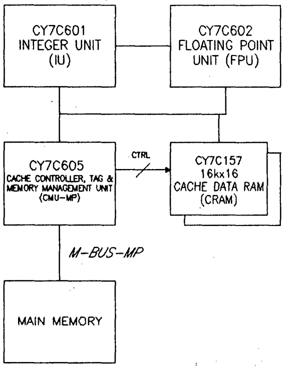


Figure 14. The SPARC Multiprocessing Cluster

The CY7C605 Cache and Memory Management Unit for Multiprocessing. The CY7C605 CMU-MP includes all of the features of the CY7C604 CMU (uniprocessing version) plus extra provisions for multiprocessing. It is fully compliant with the SPARC Reference MMU Architecture Standard. It has a 32-bit (4 GB) virtual address space and a 36-bit (64 GB) physical address space. Its memory management features include support for 4k multiple contexts, a 4 KB page size, an on-board 64 entry fully associative Translation Lookaside Buffer (TLB), support for memory address protection checking, support for hardware table walking, and support for sparse address spaces with a 3-level page table map. For cache control, the CMU contains 2K direct mapped virtual cache tag entries and support for 32 byte line size, meaning that it can manage a 64 KB direct mapped cache. It also has support for either write-through with no write allocate or copy-back with write allocate. Copy-back with write allocate poses no performance degradations since the CMU has a full 32 byte cache read buffer. The CMU is also capable of posted writes via an on-chip 32-byte write buffer, which support fully buffered STORE DOUBLEs. The advantage of this, of course, is that it increases the performance of the cache when a write miss is encountered by allowing the main memory update to occur in the background. The CMU also contains a cache lock mechanism, which allows entries to be locked in the cache thereby enabling deterministic response time for real-time applications. The CMU also provides for five levels of cache flushing. It has a 64 bit multiplexed address/data bus that provides the interface to Mbus. The CY7C605 CMU-MP provides full alias detection and correction through use of both a virtual and physical cache tag array. The addition of physical tags, which are not present in the CY7C604 CMU-UP, serves two purposes. First, this second bank of cache tags acts as a reverse translation unit, allowing on-chip detection and correction of aliasing. Second, the physical tag array permits bus snooping to occur completely independent of the processor, which interfaces to the virtual cache through the virtual cache tag array. Bus snooping is a mechanism whereby the CMU monitors all activity on the Mbus, watching for invalidation broadcasts or requests for data from other caches in the system, and responds to them. The key advantage of this second set of tag entries is that it enables the bus snooping logic to be decoupled from processor traffic, resulting in a substantial performance increase. The CMU-MP contains

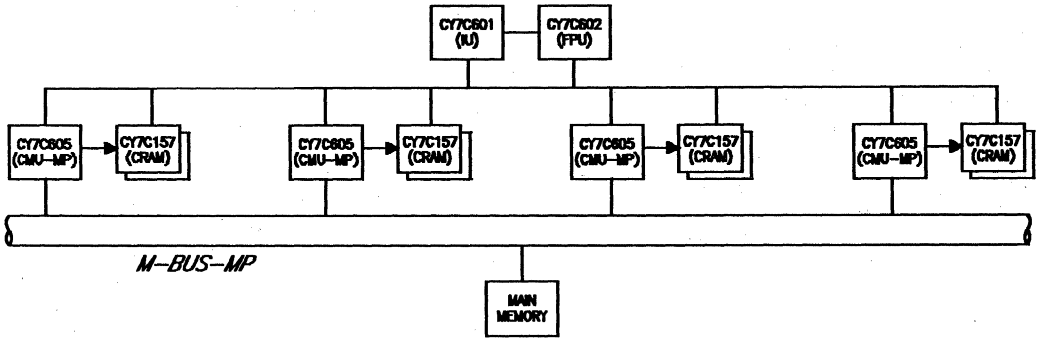


Figure 15. Fully Extended SPARC Cache

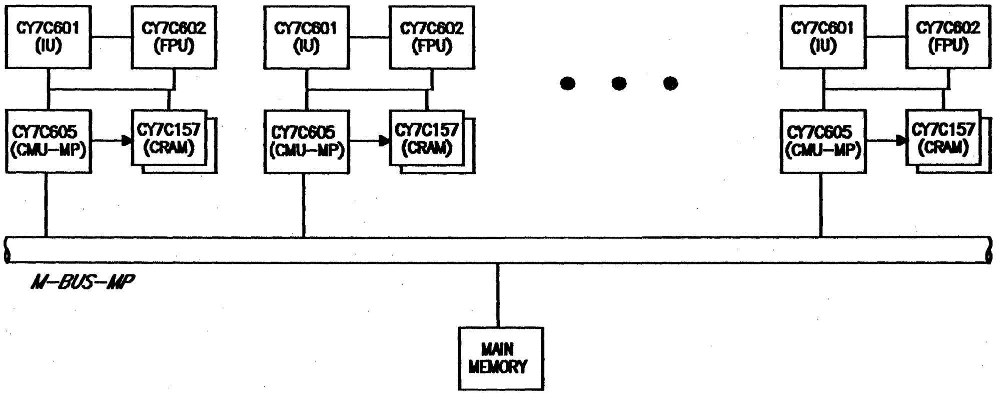


Figure 16. A SPARC Multiprocessing System

full support for the Futurebus MOESI cache consistency model. The MOESI (Modified, Owned, Exclusive, Shared, Invalid) model enables multiple caches to coexist on a single bus and share a global main memory while guaranteeing multicache consistency. Using this methodology, each entry in a cache can be in one of five states: PRIVATE CLEAN, PRIVATE DIRTY, SHARED CLEAN, SHARED DIRTY, or INVALID. If an entry is located in only one cache in the system, it will be either PRIVATE CLEAN or PRIVATE DIRTY. If more than one cache shares unmodified data, they will all be in the SHARED CLEAN state. Once a cache modifies shared data, it marks the data SHARED DIRTY, broadcasts a invalidation message informing all other caches that have that particular piece of data to mark their entries INVALID, and immediately becomes responsible for responding to any further requests for that particular piece of data. Note that any time a processor is in one of the DIRTY states, it becomes the "owner" of the data, and is responsible for servicing any requests for that data. Finally, the CMU-MP supports direct data intervention and reflective main memory. Direct data intervention provides a significant performance increase over indirect data intervention. To illustrate the difference between direct and indirect data intervention, consider a MP system with a common main memory and, for simplicity, two caches. Cache A retrieves a line of information from main memory and modifies it, thus becoming the owner of the data. At some point in the future, Cache B requests the same piece of information. In a system using indirect data intervention, Cache A would inform Cache B that it had a miss and to attempt to gain access to the bus at some later point. Cache A would then seize the bus and update main memory. Meanwhile, Cache B is spinning on the bus, trying to gain access while it's processor is on hold, awaiting the new data. When Cache A is finished updating main memory, it releases the bus. Cache B gains access, and begins to retrieve the data from main memory. Eventually, after a considerable number of cycles, processor B is released from hold and permitted to continue. In a system using direct data intervention, the data requested by Cache B would be supplied directly by cache A, resulting in considerably fewer hold cycles for processor B. Additionally, with a reflective main memory system, main memory would observe the transfer of information and update itself at the same time. With a non-reflective main memory, main memory would contain stale data relative to the cache's.

MBus. This a fully synchronous 64-bit multiplexed address/data bus that supports multiple bus masters and has a peak transfer rate of 320 MB/s at 40 MHz. All signals are sampled on rising clock edges. All signals are driven active and inactive. Mbus includes support for single address/multiple data cycle bursts in 16, 32, 64, and 128 byte sizes with full retry support. Finally, central arbitration is separate from the master and slave. The type of arbitration scheme that is used is completely up to the user. The cache consistency model for the Mbus is based on the Futurebus MOESI model.

Adding a Cache Hierarchy to SPARC MP Systems

In this section, two possible multilevel cache implementations are presented for SPARC multiprocessing systems. For highest performance, both topologies require a level 2 cache controller that is more complex than the cache controller in the CY7C605. Specifically, the level 2 cache needs to be capable of fully concurrent bus snooping and direct data intervention. In addition, it would generally be preferable that the level 2 cache have a larger line size than the level 1 cache. This means that the level 2 cache controller needs to be sector-based, thus adding to the complexity of the level 2 cache controller.

Figure 17 shows a single level cache extension topology. This topology forces the level 2 cache to manage cache consistency, which can be a performance benefit because consistency management is moved to the furthest possible point away from the processor, which tends to cause fewer hold cycles for the processor, thus increasing performance. This topology would permit smaller level 2 caches. Accordingly, if speed of the level 2 cache is critical, this topology has a definite advantage because small caches are easier to optimize for speed. The main disadvantage of this topology is that the level 2 cache is not shared by several level 1 caches, thus resulting in higher total system cost since each level 2 cache will require its own controller.

Topology 2, which is presented in *Figure 18*, is a multilayer bus-based hierarchy. This topology permits a common level 2 cache, resulting in lower cost than topology 1 since it does not contain multiple level 2 cache controllers. However, a level 2 cache size of 2 MB or more will probably be required to achieve high system-level performance. As a result of this large cache size, this topology would probably result in a slower (perhaps multicycle) level 2 cache. If cost of the level 2 cache is critical, this topology is probably the best choice.

To summarize, for performance-critical applications, a multicache hierarchy like topology 1 produces a faster second level cache but at a higher cost. For cost-sensitive applications, topology 2 is best because it results in a lower device cost, particularly for a multicycle level 2 cache.

Summary

Cache design is becoming a very common design methodology, having become commonplace even in the design of personal computers. Basic cache design parameters and techniques have been extended to multilevel hierarchies in both single-processor and multiprocessor systems, both of which introduce an additional level of complexity to traditional cache design factors. Additionally, multilevel cache hierarchies tend to make cache coherence and consistency management more complex, especially in multiprocessor systems. Cypress is taking a leading role position toward solving these problems, as evidenced by the multicache consistency features that have been designed into our SPARC family of products. The goal of Cypress Semiconductor is to allow our customers to leverage this high level of integration, making multiprocessing and multilevel cache systems realizable with the least amount of design effort possible.

Glossary of Cache Memory Terms

Arbitration time

The time taken to determine which of simultaneous contenders for a service takes priority.

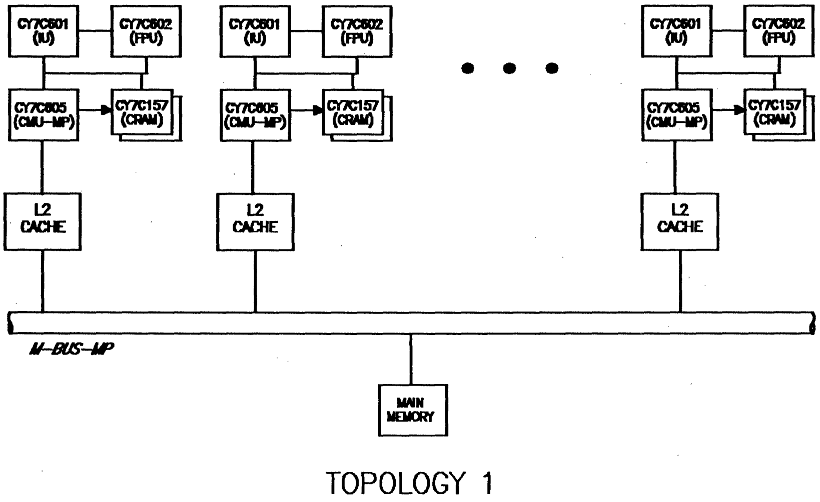


Figure 17. SPARC Single-Level Cache Extension Multilevel Cache Topology

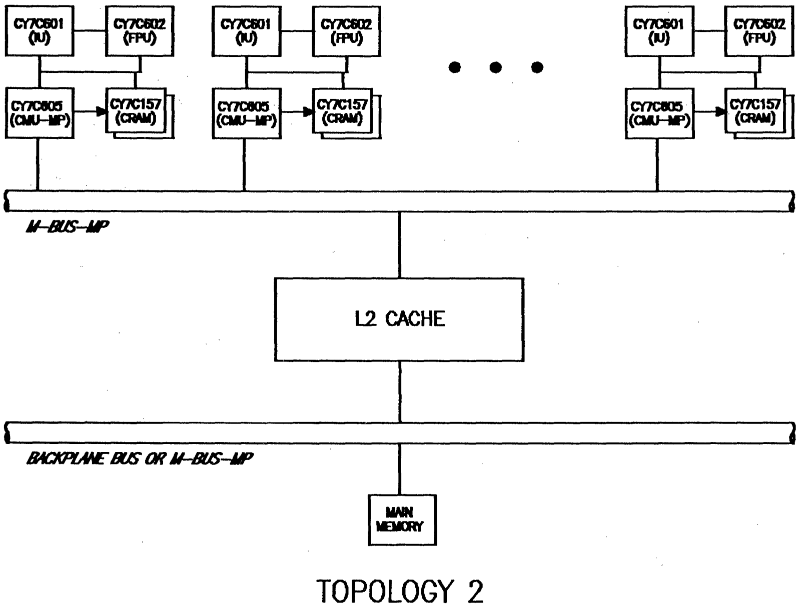


Figure 18. SPARC Bus-Based Multilevel Cache Topology

Associativity

The number of information elements per set in a cache.

Consistency (coherency)

Agreement between shared contents of members of the memory system.

Effective access time

A cache performance metric giving the average time required to service a reference.

Line (block)

The basic unit of information exchange between a cache and main memory or between a parent cache and it's child(ren) cache(s).

Miss rate

A cache performance metric giving the probability that a reference will produce a miss.

Page table

A set of tables, which are stored in main memory, that translate virtual addresses to physical addresses.

Physical address

The actual hardware address of a piece of information in main memory.

Placement algorithm

The method used to determine where a block may reside in a cache; often selects the set of a reference.

Reference

A request by the processor to read or write a memory location.

Set

A collection of cache locations in which a line may reside.

Virtual address

An address generated by a program and later translated into a real address for main memory.

References

Baer, J.L. and Wang, W.H., "On the Inclusion Properties for Multilevel Cache Hierarchies," *Proceedings of the 15th Annual Symposium on Computer Architectures*, February 1988, pp. 81-88

Gregory, Richard, "Caching Designs Eliminate Wait States to Relieve Bottlenecks," *Computer Design*, October 15, 1988, pp. 65-73

Hill, Mark D., "A Case for Direct Mapped Caches," *IEEE Computer*, December 1988, pp. 25-40

Kabakibo, Aiman, et al, "A Survey of Cache Memory in Modern Microcomputer & Minicomputer Systems," *IEEE Micro*, March 1987, pp.210-227

Pohm, A.V. and Agrawal, O.P., High Speed Memory Systems, Reston Publishing, 1983

Short, R.T. and Levy, H.M., "A Simulation Study of Two- Level Caches," *Proceedings of the 15th Annual Symposium on Computer Architectures*, February 1988, pp. 81-88

Smith, A.J., "Cache Memories," *Computing Surveys*, Vol 14, No 3, September 1982, pp. 473-530

Smith, A.J., "Cache Memory Design: An Evolving Art," *IEEE Spectrum*, December, 1987, pp. 40-44

Smith, A.J., "Line (Block) Size Choice for CPU Memories," *IEEE Transactions on Computers*, vol C-36, No. 9, September 1987, pp.1063-1075

Smith, J.E. and Goodman, J.R., "A Study of Cache Organizations and Replacement Policies," *ACM Computing Surveys*, 1983, pp. 132-137

NOTES:

CYPRESS SEMICONDUCTOR

SPARC as a Real-Time Controller

Overview of Real-Time Computing

A real-time system is one in which it is mandatory to react to external events as they happen. These systems are, by nature, event driven as they respond to external, asynchronous stimuli and must do so in a timely manner. If logical correctness, as well as timing correctness are not satisfied severe consequences will result. While the need for logical correctness is obvious, the need for timing correctness arises due to the possible physical impact of the controlling system's activities. If a computer controlling a satellite does not respond to an external event in time, the satellite may collide with a foreign object and be knocked out of its orbit.

At the highest level, a real-time system can be viewed as one which acquires data and detects the occurrence of events by means of hardware inputs. These inputs are then processed with the results being transmitted to hardware outputs. The processing of this data is the job of the embedded computer. The control of the embedded computer is the job of the real-time operating system.

When defining a real-time system it is essential to partition the functions to be performed into individual units. These units are called tasks and are implemented as software modules that can be invoked to perform a particular function. Although there are usually many tasks associated with a real-time system, there generally are a limited number of processors to execute these tasks. This paper will concentrate on the simplest case where a single processor is involved.

Since multiple tasks are competing for use of a limited resource, the processor, it is crucial that tasks be prioritized. The highest priority task that is ready to run at any given time, must actually be running. This will often lead to a case where a higher priority task becomes ready while a lower priority task is executing. In this case, the lower priority task must immediately be pre-empted and the higher priority task must take control of the processor. This is the concept of pre-emptive scheduling and is essential in all real time systems.

The real-time systems design considerations described in the previous paragraphs are the general behavior of a real-time system. In order to put this into perspective the following paragraphs will deal with specific examples.

In this particular example, the following tasks are defined in prioritized order (task 1 through task 6). Included in this system is a real time clock that will generate an interrupt to the processor every 500 microseconds. Refer to *Table* 1 for the CPU requirements of this example.

Tasks 1-5 all have specific jobs that require a fixed amount of time. Task 6, in this case, will check for user commands and vary in the amount of time needed based on whether or not a user command is present.

Table 1. CPU Requirements

Task	Duration	Operating Speed
1	35 uS	2000 Hz
2	100 uS	1000 Hz
3	1 mS	333 Hz
4	200 uS	200 Hz
5	1 mS	200 Hz

Table 2. CPU Time Per Second

Task	Time/ Invocation	Invocations	Total Time
1	35 uS	2000	70 mS
2	100 uS	1000	100 mS
3	1 mS	333	333 mS
4	200 uS	200	40 mS
5	1 mS	200	200 mS
Background	10ms	Display	
Background	200ms	Command	

By examining the above data, the following requirements of CPU time per second, for the individual tasks, are noted in *Table 2:* In this case tasks 1-5 are using 743 mS, which leaves 257 mS for the background task to execute. This means that the background task will execute at a worst case rate of 1.3 times a second. In the best case, frequency of the background task would be 25 times a second, which means the display can be updated 25 times a second, while user commands can only be processed at the rate of 1.3 per second.

In this example, the overhead associated with switching processor contexts between tasks has not been taken into account. The state of the processor at the time of pre-emption is saved with each context switch. Then, the scheduler determines the next task to run. Finally, the state of the new task is loaded into the processor. In commercially available real-time operating systems the time required for a task switch generally ranges from 25 microseconds to over 100 microseconds for some processors.

Context Switch Overhead

If, in the above example, a 25 microsecond task switch overhead is included, the following system behavior occur. *Table 3* shows how one second of CPU time breaks down.

In this case, over 14 percent of the total CPU time was spent on nothing but overhead; no useful work was done.

The frequency of the background task in this case will only run at a best case frequency of 11 times a second, while the worst case frequency will only be once every other second.

Table 3. 25 uS Context Switch Overhead

Tasks 1-5	743 mS
Switch Overhead	25 uS
Number of Switches	5733
Overhead	143 mS
Background Task	114 mS

If the context switch overhead is increased to 35 microseconds, another interesting thing associated with real-time systems occurs, as shown in *Table 4.*

Table 4. 35 us Context Switch Overhead

Tasks 1-5	743 mS
Switch Overhead	35 uS
Number of Switches	5733
Overhead	200 mS
Background Task	57 mS

Although it seems as if everything will work, critical timing parameters have been violated. In this particular case, task 5 will be scheduled to run its second time when it hasn't been allocated enough CPU time to complete its first run. An example C program to compute context switch overhead is in *Appendix A*. Context switching is only one of the many factors to take into account when designing a real-time system. Another critical factor is interrupt latency.

Interrupt Latency

The requirement to meet externally imposed deadlines is at the heart of what is termed a real-time system. Real-time computing is that type of computing where the correctness of the system depends not only on the logical result of the computation, but also on the time at which the results are produced. A system must be fast as well as predictable.

The term used to specify the predictability of the system is the worst case interrupt latency . This is defined as the maximum amount of time a system will take before responding to an external event. This parameter is usually a good indication of the worthiness of a particular processor as a real-time controller.

The interrupt latency directly effects two key factors of system performance, the guaranteed response time to an event, and the guaranteed response time of any individual task. The response time to an event can be thought of as the maximum amount of time that will elapse before the system can identify that an event has occurred and respond with the necessary action. An example would be detection of meltdown on a nuclear power plant. In this case, the processor would do the critical actions necessary to shut the reactor down from the interrupt handler without paying the time penalty of a context switch. The response time of a task is the maximum amount of time it takes to pass control from a lower priority task to a pre-empting higher priority task. *Table 5* shows the system characteristics of the effect of interrupt latency in a real-time system. Many factors contribute to interrupt latency. The processor itself has a worst case interrupt response time. The memory subsystem may also contribute to interrupt latency. The operating system may be required to disable interrupts during critical sections of code, thus adding to interrupt latency.

Table 5. Effect of Interrupt Latency

Event	Worst Case Time
Task Switch	35 uS
Interrupt latency	25 uS
Response to event	25 uS
To pre-empting task	60 uS

Interrupt response time varies between processors. Some processors are designed such that when an interrupt occurs, the entire state of the machine is saved. In this case, the interrupt handler simply starts executing without having to worry about the context of the interrupted task. While this may be convenient for the person writing the interrupt handler, it introduces sufficient overhead and slows interrupt response time. Other processors simply vector to the interrupt handler and make the interrupt routine responsible for saving any part of the state of the interrupt task that it might use. This state must then be restored upon exit from the interrupt handler. This is a good approach as no unnecessary overhead is introduced. The best approach in minimizing interrupt latency at the processor level is one where the hardware has a dedicated set of registers reserved for interrupt handlers. With this approach, the interrupt handler need not be concerned with saving and restoring the interrupted tasks working registers. Another factor that must be taken into account is the latency of the memory system. In a design using dynamic memory, included in the interrupt latency is the worst case memory cycle timing for fetching of interrupt handler instructions. In a cache system, the worst case timing would include the case of a cache miss. With the speed of processors in the 25 Mhz to 40 Mhz ranges, failure to take these things into consideration could have drastic effects.

Just as important as the time it takes to switch tasks or respond to interrupts, is the window of time during which the operating system is unable to do these things. The ability for an operating system to do a context switch in 10 microseconds isn't useful if that operating system disables context switching for periods of 50 microseconds or more while doing something else. Some of the reasons an operating system might disable interrupts would be for placing a task in a ready queue, or when accessing a critical region while doing intertask communication, resource allocation, or task synchronization.

When accessing a critical region, there must be some way of getting uninterrupted access to a shared variable. Although some processors have support for this in hardware, this is not always the case. The following is an example of the overhead involved when hardware support for uninterruptable access to shared variables is not present. The two tasks are arranged as shown in *Table 6*.

In this example, two tasks are defined. Task 1 will count the number of input pulses from an input stream. Task 2 will read the total number of pulses every second, clear the count variable, and perform a series of operations based on the total number of pulses into the system . If special care is not taken in the critical region of accessing the shared count variable, the following may occur:

Table 6. Format of Tasks

Task 1	Task 2
count- > register	count- > register
register + 1- > register	0- > count
register- > count	process based on count

1. Task 1 has control /* count is at 200 */
   count- > register
   register + 1- > register
   interrupt occurs

2. Task 2 gets control (One second has elapsed)
   count- > register
   0- > count
   execute based on count

3. Task 1 resumes
   register- > count

It is obvious that a serious problem has occurred, the variable count contains a value of 201 when it should be 1. This is a common problem that must be overcome in a multitasking environment. The key to eliminating this is uninterruptable updating of shared variables. In processors without hardware support for this, the only way to update a shared variable without the possibility of being pre-empted is by disabling interrupts. *Table* 7 shows modifications.

Table 7. Modified Format of Tasks

Task 1	Task 2
disable interrupts	disable interrupts
count- > register	count- > register
register + 1- > register	0- > count
register- > count	enable interrupts
enable interrupts	execute based on count

While this solution will work, and the maximum amount of time in which interrupts are disabled is minimal, everything is not as it seems. The major problem is that interrupts can only be disabled in supervisor mode. This means that a software trap must be executed, the processor must branch to a trap vector, change into supervisor mode, execute the few uninterruptable instructions, then go back to where it originally came from. All this time that the processor is uninterruptable must be taken into consideration when calculating worst case interrupt latency. The above proposed solution is only valid for single processor systems. In a multiprocessor system, some form of hardware lockout is essential.

SPARC as a Real-time Controller

As real-time systems vary widely in requirements, it is important that a particular processor chipset provide the flexibility to meet the needs of specific applications. It does not make sense to pay for a processor which has a built in floating point unit to do strictly integer operations. The same holds true when paying for a processor with a built in MMU when only a physical memory system is used. The Cypress SPARC chip set is specifically designed to meet the needs
of individual applications without forcing the designer to buy something he does not need. *Table* 8 shows the SPARC family of chips. These parts can be used in any combination to make up a system to fit the desired application.

Table 8. RISC 600 Family of SPARC Chips

Device	Description
CY7C601	Integer Unit
CY7C602	Floating Point Processor
CY7C604	Cache Tag-Controller/ MMU
CY7C157	Cache RAM

Worst Case Processor Interrupt Response Time

The CY7C601 has been designed to minimize interrupt latency at the processor level. The processor dedicates eight of its one hundred and thirty-six registers strictly for use by interrupt handlers. When an interrupt occurs, the interrupt routine automatically gets a new set of eight registers with which to work. On an interrupt, the processor switches to supervisor mode, gets the new set of registers, and completes execution of the first instruction in the interrupt routine in a worst case time of 14 clock cycles. At 40MHz that is 350 nanoseconds. The program counter and next program counter of the interrupted task, are saved automatically in two of these registers, with the remaining six registers at the disposal of the interrupt routine. Upon return from the interrupt, the state of the interrupted task is automatically restored by the processor, this is done in two clock cycles or 50 nanoseconds at 40MHz.

Achieving Deterministic Response Time

The CY7C604 CMU has two special features, which helps to guarantee deterministic response for systems using either virtual or physical addressing with or without cache memory. The MMU allows selected pages to be locked into the Translation Lookahead Buffer (TLB). This ensures that critical pages of memory are always in main memory and the delay associated with a table walk is not incurred. In systems using cache memory, the CY7C604 provides a feature allowing the cache to be locked. A user can load the cache with time critical code, such as interrupt handlers and time critical tasks, and be sure that these routines will always be present in the cache. With these features, memory latency is no longer a problem and predictability is guaranteed.

Semaphore Support in Hardware

Included in the instruction set of the CY7C601 are two instructions that provide uninterruptable access to an external memory location. The SWAP instruction exchanges the contents of a selected register with the contents of the addressed memory location. The atomic load-store instruction moves a byte from memory into the selected register and then rewrites the same byte in memory to all ones. Both of the instructions are executed without allowing intervening asynchronous traps. Either of the two instructions can be used to create a semaphore for accessing a critical region without the need to enter supervisor mode and disable interrupts. The SWAP instruction would be used for counting semaphores and the atomic load-store is appropriate for a simple semaphore for critical regions.

Alternate Register Models For Sparc

The Cypress CY7C601 has a total of 136, thirty two bit registers. These are divided into a set of 128 local registers and eight globals. The way these registers are used is configurable by accessing a processor register called the Current Window Pointer (CWP). Two common models supported by commercially available compilers and operating systcms are the standard register windowing model optimized to minimize procedure call overhead, and an alternate model used to significantly reduce the time required for a context switch.

Register Windowing Model

In this mode of operation, the register file is divided up into a set of eight overlapping register windows. Each window contains a set of twenty-four local registers. The registers in each window are divided into three sets of eight registers referred to as INS, LOCALS, and OUTS. At any given time, only one window and the eight globals are accessible to the processor. The windows are joined together in a circular stack with each window sharing its INS and OUTS with adjacent windows. Two instructions are provided for rotating the windows between procedures.

A "save" instruction is used with a procedure call to allocate the next window for the called procedure. Before executing the save instruction, the calling procedure would store the parameters to be passed in its OUT registers. Upon execution of the save instruction, the register set would be rotated such that the called procedure would have access to the passed parameters in its IN registers.

A "restore" instruction is used with a return from procedure to restore the register set of the calling procedure. Before executing the restore instruction, the called procedure would store the parameters to be returned to the calling procedure in its IN registers. Upon execution of the restore instruction, the register set will be rotated back to its previous position with the returned parameters in the callers OUT registers.

Because the processor logically provides new LOCALS and OUTS with each procedure call, local register values need not be saved and restored across calls. The overlapping registers also minimize the overhead of passing and returning procedure parameters as they are passed in registers as opposed to the main memory stack.

Fast Task Switch Register Model

In this mode of operation, the register set is divided into four non-overlapping sets of twenty four registers. Three of the four register sets are dedicated to the three highest priority or time dependent tasks. The last set of registers is shared between all remaining tasks. Associated with each set of registers are a set of eight independent registers to be used by interrupt handlers. These registers are also used to store the state of the processor on a task switch.

Using this register model, a task switch to any of the three highest priority tasks can be done in under a microsecond. A task switch to one of the other tasks can be done in less than three microseconds.

When an interrupt occurs, the processor automatically switches register sets to access the interrupt registers corresponding to that particular task. If the interrupt initiates a task switch, the state of the processor is saved in the interrupt registers. If the new task is one of the three high priority tasks, its state is loaded from its dedicated interrupt registers and execution begins immediately. In this case, the state of the machine is merely the PSR, PC, NPC and possibly a few other control registers. The general purpose registers have not been affected as they are dedicated to this task.

If the new task is one which shares a set of registers, the state of the task previously using that register set is saved to memory and the state of the new task is loaded into the processor. In this case the state includes the minimal processor state as well as the twenty-four general purpose registers.

Example 1 --- Switching to a higher priority task

1)Interrupt occurs
  Automatically switches to interrupt registers
  PC and NPC saved in interrupt registers
2) Save PSR and any other control register to interrupt registers
3) Load the pointer to the new tasks interrupt registers into the CWP
4) Restore new tasks PSR and any other control registers
5) Execute RETT (return from trap)

Example 2 -- Switching to a lower priority task

1) Interrupt occurs
  Automatically switches to interrupt registers
  PC and NPC saved in interrupt registers
2) Save PSR and any other control registers to interrupt registers
3) Load pointer to the shared set of working registers into the CWP
4) Save the registers to memory
  (these are the registers of the previous task using the window)
5) Restore the working registers of the new task from memory
6) Update the CWP to point to the shared tasks interrupt registers
7) Save the eight interrupt registers containing the state of the previous task running out of these registers to memory
8) Restore the state of the new task
9) RETT (return from trap)

As can be seen, each register model has certain advantages. Using register windowing significantly reduces both procedure call overhead and data bus traffic as parameters are passed in registers. This also has the affect of caching local variables as each procedure get a new set of local registers. The price paid for this is in the context switch overhead. In this case, all of the used registers, i.e. up to 136, as determined by the Window Invalid Mask (WIM), a processor status register, must be saved and restored on a context switch. When using the fast context switch register model, one does not get the benefit of register windowing i.e., ultra-fast procedure calls. But, in return, gets the benefit of four separate register files and very fast context switching. In this mode, parameters are simply passed on the stack as is done on most other architectures. Each task is allocated twenty-four general purpose local registers and eight global registers which is the same as the total number of registers in most other architectures.

Since the usage of the registers on the CY7C601 are configurable by software, these modes can also be mixed, allowing the benefits of both models. The SPARC register set as well as the entire Cypress chip set has been designed to cover a wide range of applications efficiently.

Appendix A. Sample C Program to Computer Context Switch Overhead

```
/*****************************************************************************************************************/
/*                                                                                                               */
/* This program is used for determining the overhead of context switching in a real-time system. This simulation does not take into */
/* account interrupt latency, memory latency, or any of the other many possible forms of overhead associated with a realtime      */
/* system, but these can easily be added. The current version should be sufficient to give a good idea of how much time the kernel */
/* is spending on context switching.                                                                             */
/*****************************************************************************************************************/
#include \c\ms\include\math.h
#include \c\ms\include\stdio.h
#define BCKGRND 100
FILE *fp; int openfile; char fname[35]; int numtasks;
main (argc, argv)
int argc;
char *argv[];
{
int         i,j;
int         iterations;
int         curr_task;
int         time[100];
int         duration[100];
int         frequency[100];
int         total;
float       background;
int         swtime;
int         switchh;
int         temp;
int         temp1;
int         sampfreq;
float       tempflt;
float       cs_time;

create_file();
temp = 0;

/* get number of simulation points per second */
while (temp= =0)
    {
    place (7,4,"Enter the sampling rate in Hz (100 - 10000) : ");
    locate (7,58);
    ceol();
    iterations = 0;
    temp = getchar();
    while (temp< >0xa)
            {
            if ((temp > = 0x30) && (temp < = 0x39))
                    {
                    temp = temp - 0x30;
                    iterations = iterations * 10;
                    iterations = iterations + temp;
                    }
            temp = getchar();
            }
    temp = 1;
    locate (22,5);
```

Appendix A. Sample C Program to Computer Context Switch Overhead (continued)

```
     ceol();
     if (((iterations % 100) != 0) || (iterations  =100) || (iterations  =10000))
               {
               temp = 0;
               place (22,5,"Error must be = 100 or 10,000 and a mult of 100");
               }
     },
/* time in microseconds of one clock tick */
sampfreq = 10000 / (iterations / 100);

place (8,4,"Enter the context switch overhead in microseconds : ");
locate (8,58);
swtime = 0;
temp = getchar();
while (temp< >0xa)
     {
     if ((temp > = 0x30) && (temp< = 0x39))
               {
               temp = temp - 0x30;
               swtime = swtime * 10;
               swtime = swtime + temp;
               }
     temp = getchar();
     },

temp = 0;
while (temp= =0)
     {
     place (9,4," Enter the number of tasks (100 max) : ");
     locate (9,58);
     ceol();
     numtasks = 0;
     temp = getchar();
     while (temp < > 0xa)
               {
               if ((temp > = 0x30) && (temp < =  0x39));
                         {
                         temp = temp - 0x30;
                         numtasks = numtasks * 10;
                         numtasks = numtasks + temp;
                         }
               temp = getchar();
               }
               temp = 1;
               locate (20,5);
               ceol();   if (numtasks  >100)
                         {
                         temp = 0;
                         place (20,5 ,"Maximum number of tasks is 100");
                         }
               },
/* tasks numbered 0 to n */
numtasks = numtasks - 1;
for (i=0; i=numtasks; i++)
```

Appendix A. Sample C Program to Computer Context Switch Overhead (continued)

```
        {
        temp = 0;
        while (temp= =0)
                {,
                locate (i+11,4);
                printf("Enter the frequency of task %d in Hz",i);
                locate (i+11,60);
                ceol();
                frequency[i] = 0;
                temp = getchar();
                while (tem< >0xa)
                        {
                        if ((temp > = 0x30) && ( temp  < =0x39))
                                {
                                temp = temp - 0x30;
                                frequency[i] = frequency[i] * 10;
                                frequency[i] = frequency[i] + temp;
                                }
                        temp = getchar();
                        }
                locate (20,5);
                ceol();
                locate (21,5);
                ceol();
                if (frequency[i] 0)
                        {
                        if ((iterations % frequency[i]) != 0)
                                {
                                locate (20,5);
                                printf (" Warning : %d and the simulator frequency : %d are not multiples",frequency[i],iterations);
                                place (21,5," Would you like to re-enter the value (not mandatory) (y/n) : ");,
                                locate (21,70);
                                temp = getchar();
                                temp1 = getchar();
                                /* CR */
                                if ((temp= ='Y') || (temp= ='y'))
                                        temp = 0;
                                else
                                        temp = 1;
                                }
                        }
                else
                        {
                        place (20,5," Frequency must be greater than zero ");
                        temp = 0;
                        }
                }
        /* frequency[i] will be used with modulo operator to see when task ready */
        frequency[i] = iterations / frequency[i];
        /* integer divide */
        }
    locate (20,5);
    ceol();
    locate (21,5);
```

Appendix A. Sample C Program to Computer Context Switch Overhead (continued)

```
ceol();
for (i=0; i=numtasks; i++)
    {
    locate (i+numtasks+14,4);
    printf("Enter the duration of task %d in microseconds",i);
    locate (i+numtasks+14, 60);
    duration[i] = 0;
    temp = getchar();
    while (temp< >0xa)
            {
            if ((temp > = 0x30) && (temp < = 0x39))
                    {
                    temp = temp - 0x30;
                    duration[i] = duration[i] * 10;
                    duration[i] = duration[i] + temp;
                    }
            temp = getchar();
            }
    }
/* init ialize current task */
curr_task = BCKGRND;
/* init current task, task switch needed for 1st task background task time of execution */
background = 0;
/* number of context switches */
switchh = 0;
/* init total time left in this time s lice */
total = 0;
/* check to see whether a disk file is to be opened */
    if (openfile= =1)
    init_file();
cls();
/* init time spent in individual tasks */
for (i=0; i=numtasks; i++)
    time[i] = 0;
/* iterations start at 0 */
iterations = iterations - 1;
/* main simulation loop */
for (j=0; j=iterations; j++)
    /* number of samples */
    {
    /* screen oup ut to show system didn't die */
            if ((j % 100)= =0)
            {
            locate (10,6);
            printf (" Doing simulation loop %d of %d ", j,iterations+1);
            }
    total = total + sampfreq;
    /* increment clock time for each time slice scheduling of tasks */
    for (i=0; i=numtasks; i++)
            {
            /* check if task is scheduled to execute */
            if ((j % frequency[i])= =0)
                    /* modu lo operator */
                    if (time[i]= =0)
```

Appendix A. Sample C Program to Computer Context Switch Overhead (continued)

```
          /* has it completed from last time */
          {
          time[i] = duration[i];
          /* init time slice required */
          if (openfile = = 1)
                    fprintf(fp,"task%d is ready \n",i);
          }
else
/* hasn't completed previous scheduled time */
          {
          cls();
          locate (10,6);
          printf (" Need a faster processor, check simulation file");
          if (openfile= =1)
                    fprintf (fp," task %d has been scheduled again but has not completed",i);
          goto p1; /* abort simulation */
          }
}
/* print which clock tick in file */
if (openfile= =1)
          fprintf(fp,"%d:",j);
/* executing of tasks */
for (i=0; i=numtasks; i++)
          /* check for tasks 0 to n being ready */
          {
          if (total >0)
                    /* check if there is time to run the task */
                    {
                    if (time[i]>0)
                              /* is this particular task ready to run */
                              {
                              /* does a context switch actually take place or was */
                              if (i != curr_task)
                                        {
                                        total = total - swtime;
                                        /* context switch time */
                                        switchh = switchh + 1;
                                        /* # of context switches */
                                        }
                              /* can task time slice be completed */
                              if (total = time[i])
                                        {
                                        if (openfile= =1)
                                                  fprintf(fp,"%d",time[i]);
                                        total = total - time[i];
                                        /* time left in slice */
                                        time[i] = 0;
                                        /* update ready list */
                                        curr_task = i;
                                        /* mark as last task to run */
                                        }
                              /* can run portion of task */
                              else
                                        {
```

Appendix A. Sample C Program to Computer Context Switch Overhead (continued)

```
                                        /* use remaining time available in simulation slice */
                                        if (total 0)                                            {
                                                /* time still required by the task */
                                                time[i] = time[i] - total;
                                        total = 0;
                                        /* time slice has expired */
                                        }
                        /* mark in sim file that a context switch has started  for */
                        /* one task but a higher priority task has become ready */
                        /* and will has pre-empted the scheduled task     */
                        else
                                {
                                if (openfile= =1)
                                        fprintf(fp,"X");
                                }
                        /* mark state of processor */
                        curr_t ask = i;
                        }
                }
        else
                {
                if (openfile  = = 1)
                        fprintf(fp,"-");
                }
        }
else
        {
        if (openfile=  =1)
                fprintf(fp,"-");
        }
}
/* background */
/* if time left after all scheduled tasks have run, let bac kground task run */
if (total 0)
        {
        /* check to see if background was last to use the processor */
        if (curr_task != BCKGRND)
                {
                switchh = switchh + 1;
                total = total - swtime;
                }
        curr_task = BCKGRND;
        /* set curr_task to background */
        if (total 0)
                {
                if (openfile= =1)
                        fprintf(fp,"%d",total);
                /* add to background task exec ution time */
                background = background + total;
                total = 0;
                /* background takes all remaining time */
                }
        else
                {
```

Appendix A. Sample C Program to Computer Context Switch Overhead (continued)

```
                                if (openfile= =1)
                                        fprintf(fp,"X");
                                }
                        }
                else
                        {
                        if (openfile= =1)
                                fprintf(fp,"-");
                        }
                if (openfile= =1)
                        fprintf(fp,"\n");
                }
        /* screen output */
        cls();
        for (i = 0; i=numtasks; i+ +)
                {
                tempflt = (((float)iterations + 1) / (float)frequency[i]) * (float)duration[i];
                tempflt = tempflt / 1000;
                locate (i+4,6);
                printf (" Total execution time for task %d : %6.2f ms",i,tempflt);
                }
        locate (numtasks + 6,6);
        printf (" There were %d context switc hes",switchh);,
        cs_time = ((float) swtime * (float) switchh) / 1000;
        locate (numtasks + 8,6);
        printf (" Context switch ove rhead : %6.2f  ",cs_time);
        locate (numtasks + 10,6);
        printf ("Time available for background tasks : %6.2f ",background / 1 000);
        p1: locate (numtasks + 15,6);
        printf ("For more info look at simulation file ");
        locate (22,1);
        if (openfile= =1)
                fclose(fp);
        }
}
/* screen utilities supported with ansi.sys clear screen utility */
cls ()
{
printf ("%c[ 2J",27);
}
ceol ()
{
printf ( "%c[K",27);
}
locate (row,col)
int row,col;
{
printf("%c[%d;%dH",27, row,col);
}
place (row,col,text)
int row,col; char text[];
{
locate (row,col);
puts (text);
```

Appendix A. Sample C Program to Computer Context Switch Overhead (continued)

```
}
create_file()
{
int temp,i;
openfile = 0;
for (i=0;i#;i++)
      fname[i] = 0;
cls();
place (5,4,"Enter file to be created (Retu rn for no file): ");
locate (5,51);
i = 0;
temp = getchar();
if (temp=0xa)
      {
      temp = getchar();
      while (temp< >0xa)
              {
              fname[ i] = temp;
              i++;
              temp = getchar();
              }
      fp = fopen(fname,"w");
      openfile = 1;
      }
}
init_file()
{
int i;
fp rintf(fp,"\n\n\n\n Simulation Results \n\n\n\n\n");
fprintf(fp,"      Tick        ");
for (i=0; i=numtasks; i++ )
      fprintf(fp,"task%d    ",i);
fprintf(fp,"background      \n\n");
},
```

CYPRESS
SEMICONDUCTOR

Using the CY7C330 as a Multi-channel Mbus Arbiter

Introduction

This application note discusses the use of the CY7C330 as a bus arbiter for a Cypress SPARC CY7C600 RISC processor Mbus system. The Cypress CY7C330 is a high-speed synchronous Erasable Programmable Logic Device (EPLD) optimized for Finite State Machine (FSM) applications. The Cypress SPARC system utilizes a CY7C601 33MHz RISC processor, a CY7C602 Floating Point Unit (FPU), four CY7C604 Cache Controller and Memory Management Units (CMU) and eight CY7C157 16K x 16 cache RAMs make up a 256 Kb cache. The arbiter uses a combination of techniques to resolve Mbus access contention for a system with four CMU bus masters. Refer to *Figure 1* for a block diagram of the Mbus system.

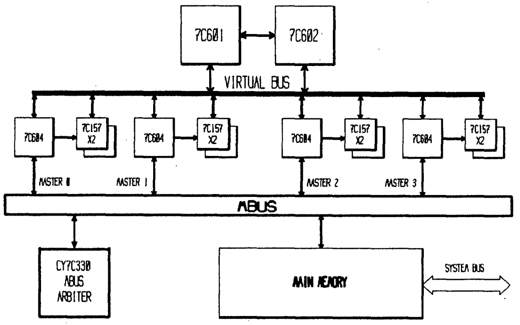


Figure 1. Mbus System Block Diagram

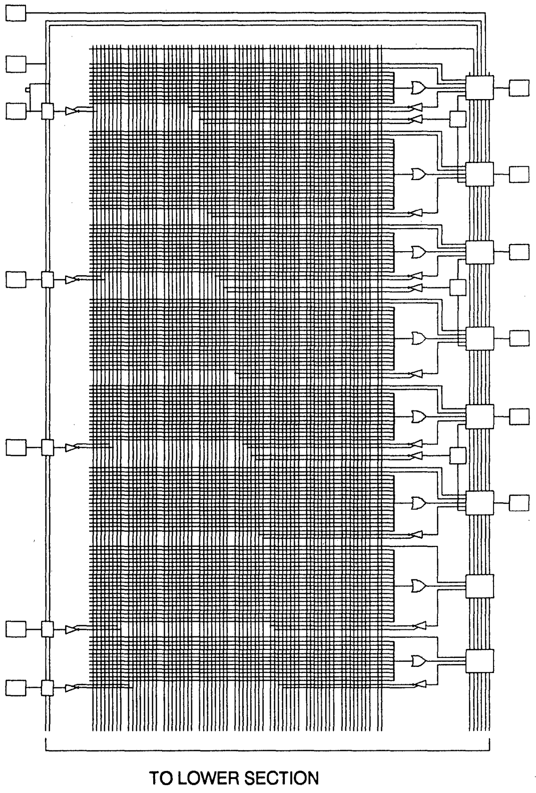


Figure 2a. CY7C330 Block Diagram (Upper Half)

CY7C330 Brief Description

The CY7C330 is a 66 MHz, high-performance PLD with 11 input latches, 17,000 programmable bits, 4 buried state registers and 12 user-configurable output macrocells. It is manufactured using a CMOS 0.8

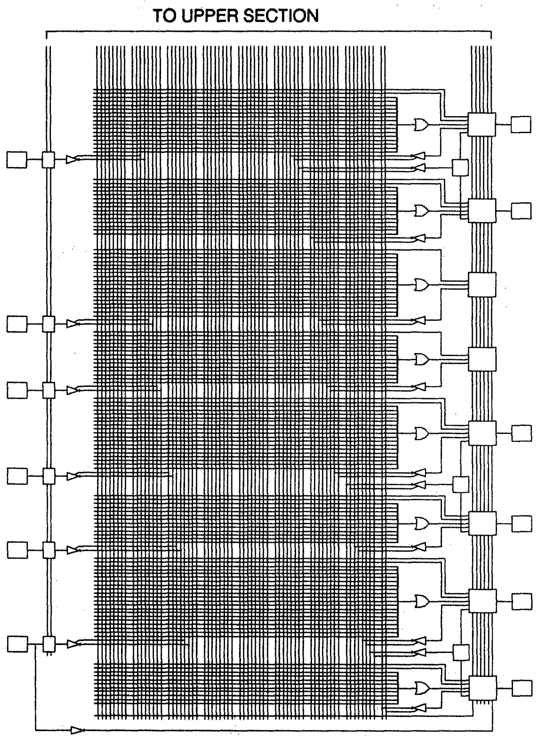


Figure 2b. CY7C330 Block Diagram (Lower Half)

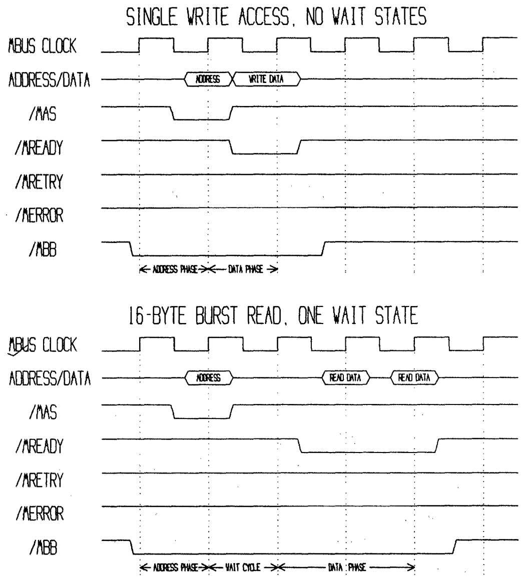


Figure 4. Mbus Data Transfer Waveforms

micron, double metal processing technology that is UV erasable and is packaged in a 28-pin 300 mil Dual Inline or LCC/PLCC package(s). It can be partitioned into multiple functional blocks as shown in this application. The CY7C330 block diagram is shown in *Figures 2a* and *2b*.

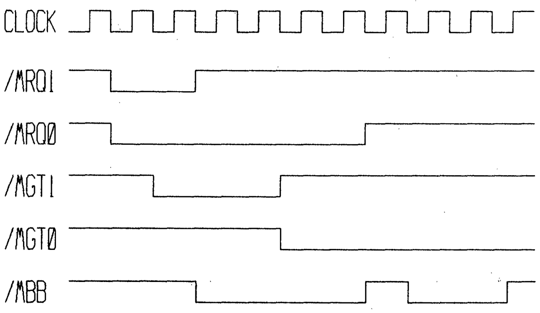


Figure 3. Mbus Multiple Request Sequence

Mbus Description

Mbus is a system bus which is defined to be a SPARC standard main memory interface for the Cypress SPARC Cache/Memory Management Unit device (the CY7C604). The "M" in Mbus stands for module and emphasizes the multi-processor module support that SPARC offers. It is a high-speed synchronous, 64-bit multiplexed address/data bus that operates at the clock rate of the CY7C601. Mbus accesses are initiated by a MASTER and responded to by a SLAVE. Generally a bus transaction takes place between a MASTER and main memory, but in the case of direct data intervention, transactions can occur between MASTERs. The handshake between the 7C604 CMMU and the arbiter consists of a request line (MRQ0-3) and a grant line (MGT0-3) for each master. A busy line (MBB) is common to all masters and indicates that the bus is in use. Refer to *Figure 3* for the multiple Mbus request sequence. By design, bus mastership and resolution of multiple requests are performed outside of the realm of Mbus and SPARC. This allows the designer to imple-

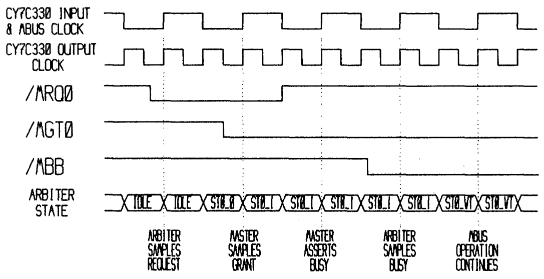


Figure 5. CY7C604 & CY7C330 Timing for Master 0

ment any arbitration scheme that best fits the system requirements. This application example describes only one such implementation.

Mbus transfers are synchronous with respect to the system clock. The data transactions across the bus consist of a single clock period address phase, and a multiple clock period data phase. Data is transferred in word (64-bit), multi-word burst, or atomic load store formats. All signals are valid and sampled on the rising edge of the system clock. The address phase is validated by the Memory Address Strobe (/MAS) signal and denotes the start of the actual data transfer. Bus states are indicated by three status lines and convey the current bus operation as well as error status. Refer to *Figure 4* for Mbus data transfer waveforms.

Timing Considerations

To meet the timing specifications of Mbus, the arbitrator must be capable of: (1) accepting a request; (2) resolving access contention (if any); and (3) granting bus rights to a master in a single Mbus clock cycle. In this application, the arbiter, a 66 MHz CY7C330, will have its input registers running at the same clock rate (33 MHz) as the CY7C601 and CY7C604s. This allows the arbiter inputs to meet the timing requirements of the Mbus masters. The output registers (including the state machine) are clocked at twice the rate of the bus masters, (66 MHz) enabling the arbiter to sample requests with the input latches on the rising edge of one Mbus clock cycle, transfer from one state to another, and grant access before the next rising edge of the Mbus clock.

The timing relationship between Master 0 (CY7C604 at 33 MHz) and the 66 MHz CY7C330 arbiter are shown in *Figure 5.*

Arbitration Scheme

With the arbitration function left to the designer, there are several resolution techniques that can be employed. Fixed priority, rotating priority, least recently used, and random priority are all contention resolution schemes that have been employed successfully, with each having their own faults. A fixed priority, for instance, will favor one requester more than the others. Rotating priority will provide a simple but not always fair approach to arbitration. An LRU arbitration scheme represents the fairest form of contention resolution but requires a highly complex implementation. The random technique will not allow predictable arbitration results and could result in performance problems.

Since there are negatives associated with most arbitration techniques, a combination of methods will be used to minimize the associated problems. In this example, we have chosen to employ both a random and a fixed priority scheme. The random scheme uses a two bit counter that increments every clock cycle and varies the priority accordingly. The priority function can be set to allow the processor to define which master has the highest priority by loading a value into the 7C330 via a store instruction. The interface to the processor accomplishing this store function requires a latched and decoded chip select along with the latched write enable connected directly to the arbiter. This priority function can be of value if critical data required by a program is being fetched from main memory by the pre-set highest priority Mbus master. The remaining channels follow a pre-set priority that is defined in *Table 1* below.

The Random Priority Counter also uses the same priority scheme and is effect only when the latched priority is disabled.

Table 1. Mbus Channel Priorities

Latched	PRIORITY			
Value	FIRST	2ND	3RD	LOWEST
11	MASTER3	MASTER2	MASTER1	MASTER0
10	MASTER2	MASTER1	MASTER0	MASTER3
01	MASTER1	MASTER0	MASTER3	MASTER2
00	MASTER0	MASTER3	MASTER2	MASTER1

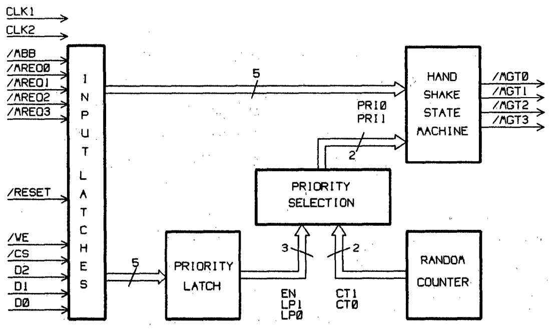


Figure 6. Arbiter Block Diagram

Design Partitioning

The design is partitioned into four functional blocks that are designed separately. Refer to *Figure 6* for the arbiter block diagram. The first block is the priority latch, which is a synchronous register using decoded and latched /CS and /WE from the CY7C601 for an enable signal. It accepts three data lines from the processor bus (one for the priority enable and two for the value of the high priority bus master) and loads the values into the dedicated registers. The random counter is a minor portion of the design, it is a free running counter that supplies a two bit binary value which is routed to the priority select block. The count changes every output clock (CLK1) cycle and provides a "seed" for the random priority function. The priority select block chooses between the priority latch outputs (LP0-1) and the random counter value (CT0-1) using the EN signal as the selection criteria. The two outputs (PRI0-1), are fed to the hand shake state machine and are used to arbitrate between bus masters when more than one simultaneous request occurs. This state machine monitors the request (MRQ0-3) and the busy (MBB) inputs and generates the grant (MGT0-3) signals that give an Mbus master ownership of the bus.

Priority Latch, Select and Random Counter Implementation

As described previously, the priority latch is a synchronous register that is loaded by the processor. The active low write enable (/WE) and chip select (/CS) are used to gate the three data bits from the bus to the three macrocells dedicated for the Priority Latch. When both /WE and /CS are active (low), the latch loads. When either are inactive (high), the output value of each register is continuously reloaded every clock cycle, thus retaining the proper value. The equations for the priority latch are shown below:

```
EN=   <oe>
      <sum>        /CS*/WE*D2
                   +EN*WE
                   +EN*CS;
LP1=  <oe>
      <sum>        /CS*/WE*D1
                   +LP1*WE
                   +LP1*CS;
LP0=  <oe>
      <sum>        /CS*/WE*D0
                   +LP0*WE
                   +LP0*CS;
```

The random counter is simply a two bit counter that changes state every output clock (CLK1) transition. It is cleared when /RESET is low and counts in a 0-1-2-3 sequence. Shown below are the equations for the random counter:

```
CT1=  <oe>
      <sum>        CT1*/CT0
                   +/CT1*CT0;

CT0 = <oe>
      <sum>        /CT0;
```

Selection between the priority latch and the random counter is done by the priority selection block. This block is a registered multiplexer that loads its register outputs with either the priority latch value if EN = 1, or the current state of the counter if EN = 0. The outputs are updated every clock and are fed to the handshake state machine.

Handshake State Machine

The Mbus handshake and arbitration are controlled by this state machine. There are thirteen discrete states that the machine cycles through in performing its function. On power-up or reset, the FSM enters the "idle" state, waiting for a bus request. Once a request (/MRQ0 for instance) has been received, the machine enters a "wait" mode (state GT0_0). In wait mode the arbiter looks for busy (/MBB) to go inactive (if not already inactive), while driving the /MGT0 output active. When /MBB goes inactive, it goes to state GT0_1 and holds /MGT0 active while waitng for /MBB to be asserted by the granted master. When /MBB is detected, the machine goes to state GT0_WAIT and looks for another request. The MGT0 grant line is held active during and after the sequence, allowing the master to maintain bus ownership until it is requested by another master.

Refer to *Figure 7* for the bus master 0 state diagram and the request/grant handshake. The operation is the identical for each of the four bus masters.

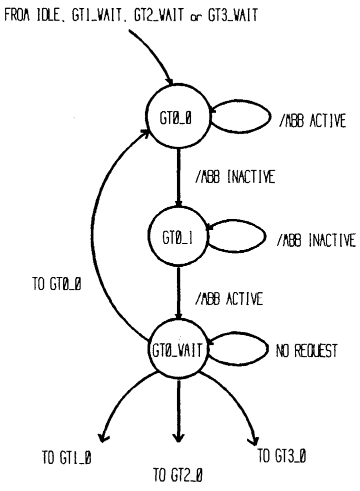


Figure 7. Bus Master 0 State Diagram.

The equations for the handshake state machine were produced from a state transition table that also included the priority encoding for the arbiter. The table was then reduced down to a manageable number of minterms using a public domain optimizer called McBOOLE[1]. Refer to *Appendix A* for the state transition table. The sum-of- products format equations were then merged into the PLD ToolKit design file with the priority latch, random counter and priority selection equations. The PLD ToolKit Design File can be found in *Appendix B.*

Design Verification

The entire CY7C330 FOUR CHANNEL MBUS ARBITER design was entered and verified using the CYPRESS PLD Toolkit. Design verification was performed using the PLD Toolkit's interactive simulator. The circuit stimuli was created using a mouse with pop down menus and drawing the waveform on the graphics screen for a each node or pin on the device. The SIMULATE command is then selected and the response waveforms are visually inspected giving the designer a very high degree of confidence in the functionality of the device before programming a part.

[1]McBOOLE, From "McBOOLE: A New Procedure For Exact Logic Minimization", M.R. Dagenias, V.K. Agarwal, N.C. Rumin, IEEE transactions on CAD of Circuit and Systems, vol. CAD-5, N.1, January 1986, p.229.

Appendix A. Mbus Handshake/Arbiter State Transition Table.

```
/*STATE TABLE FOR MBUS ARBITER HANDSHAKE STATE MACHINE -names:
MBB,MRQ3,MRQ2,MRQ1,MRQ0,PRI1,PRI0,ST3,ST2,ST1,ST0,MGT3,MGT2,MGT1,MGT0; input
ST3,ST2,ST1,ST0,MGT3,MGT2,MGT1,MGT0; output
*/
            /*RESENT                 NEXT
            STATE                    STATE
            (INPUTS)                 (OUTPUTS)
--------------------------------------------------------------------
            MMMMPP  MMMM              MMMM
            MRRRRRRRSSSSGGGG         SSSSGGGG
            BQQQQ11TTTTTTTT          TTTTTTTT
            B32101032103210          32103210*/
            X1111XX00000000          00000000        /*WAIT FOR MRQx */
            X1110XX00X0XXXX          01000001        /*GOTO GT0 */
            X1101XX00X0XXXX          01000010        /*GOTO GT1 */
            X1011XX00X0XXXX          10000100        /*GOTO GT2 */
            X0111XX00X0XXXX          10001000        /*GOTO GT3 */
            X00000000X0XXXX          01000001        /*GOTO GT0 */
            X00010000X0XXXX          10001000        /*GOTO GT3 */
            X00100000X0XXXX          01000001        /*GOTO GT0 */
            X00110000X0XXXX          10001000        /*GOTO GT3 */
            X01000000X0XXXX          01000001        /*GOTO GT0 */
            X01010000X0XXXX          10001000        /*GOTO GT3 */
            X01100000X0XXXX          01000001        /*GOTO GT0 */
            X10000000X0XXXX          01000001        /*GOTO GT0 */
            X10010000X0XXXX          10000100        /*GOTO GT2 */
            X10100000X0XXXX          01000001        /*GOTO GT0 */
            X11000000X0XXXX          01000001        /*GOTO GT0 */
            X00000100X0XXXX          01000010        /*GOTO GT1 */
            X00010100X0XXXX          01000010        /*GOTO GT1 */
            X00100100X0XXXX          01000001        /*GOTO GT0 */
            X00110100X0XXXX          10001000        /*GOTO GT3 */
            X01000100X0XXXX          01000010        /*GOTO GT1 */
            X01010100X0XXXX          01000010        /*GOTO GT1 */
            X01100100X0XXXX          01000001        /*GOTO GT0 */
            X10000100X0XXXX          01000010        /*GOTO GT1 */
            X10010100X0XXXX          01000010        /*GOTO GT1 */
            X10100100X0XXXX          01000001        /*GOTO GT0 */
            X11000100X0XXXX          01000010        /*GOTO GT1 */
            X00001000X0XXXX          10000100        /*GOTO GT2 */
            X00011000X0XXXX          10000100        /*GOTO GT2 */
            X00101000X0XXXX          10000100        /*GOTO GT2 */
            X00111000X0XXXX          10000100        /*GOTO GT2 */

            X01001000X0XXXX          01000010        /*GOTO GT1 */
            X01011000X0XXXX          01000010        /*GOTO GT1 */
            X01101000X0XXXX          01000001        /*GOTO GT0 */
            X10001000X0XXXX          10000100        /*GOTO GT2 */
            X10011000X0XXXX          10000100        /*GOTO GT2 */
            X10101000X0XXXX          10000100        /*GOTO GT2 */
            X11001000X0XXXX          01000010        /*GOTO GT1 */
            X00001100X0XXXX          10001000        /*GOTO GT3 */
            X00011100X0XXXX          10001000        /*GOTO GT3 */
```

Appendix A. Mbus Handshake/Arbiter State Transition Table.

X00101100X0XXXX	10001000	/*GOTO GT3 */
X00111100X0XXXX	10001000	/*GOTO GT3 */
X01001100X0XXXX	10001000	/*GOTO GT3 */
X01011100X0XXXX	10001000	/*GOTO GT3 */
X01101100X0XXXX	10001000	/*GOTO GT3 */
X10001100X0XXXX	10000100	/*GOTO GT2 */
X10011100X0XXXX	10000100	/*GOTO GT2 */
X10101100X0XXXX	10000100	/*GOTO GT2 */
X11001100X0XXXX	01000010	/*GOTO GT1 */
		/*CH 0 STATES */
0XXXXXX01000001	01000001	/*GT0_0, WAIT ON MBB=1 IN GT0_0*/
1XXXXXX01000001	00010001	/*GT0_0, GOTO GT0_1 */
1XXXXXX00010001	00010001	/*GT0_1, WAIT ON MBB=0 */
0XXXXXX00010001	00100001	/*GT0_1, GOTO GT0_WAIT */
X1111XX00100001	00100001	/*GT0_WAIT */
		/*CH 1 STATES */
0XXXXXX01000010	01000010	/*GT1_0, WAIT ON MBB=1 IN GT1_0*/
1XXXXXX01000010	00010010	/*GT1_0, GOTO GT1_1 */
1XXXXXX00010010	00010010	/*GT1_1, WAIT ON MBB=0 */
0XXXXXX00010010	00100010	/*GT1_1, GOTO GT1_WAIT */
X1111XX00100010	00100010	/*GT1_WAIT */
		/*CH 2 STATES */
0XXXXXX10000100	10000100	/*GT2_0, WAIT ON MBB=1 IN GT2_0*/
1XXXXXX10000100	00010100	/*GT2_0, GOTO GT2_1 */
1XXXXXX00010100	00010100	/*GT2_1, WAIT ON MBB =0 */
0XXXXXX00010100	00100100	/*GT2_1, GOTO GT2_WAIT */
X1111XX00100100	00100100	/*GT2_WAIT */
		/*CH 3 STATES */
0XXXXXX10001000	10001000	/*GT3_0, WAIT ON MBB=1 IN GT3_0*/
1XXXXXX10001000	00011000	/*GT3_0, GOTO GT3_1 */
1XXXXXX00011000	00011000	/*GT3_1, WAIT ON MBB =0 */
0XXXXXX00011000	00101000	/*GT3_1, GOTO GT3_WAIT */
X1111XX00101000	00101000	/*GT3_WAIT */

Appendix B. PLD ToolKit Source File for Mbus Arbitrater

```
CY7C330;

                    {DESIGN FILE: FOUR CHANNEL MBUS ARBITRATION UNIT WITH
                    RANDOM PRIORITY COUNTERS AND SYNCHRONOUS PRIORITY ENABLE}
CONFIGURE;

                    {INPUTS}

CLK1,               {Output Clock 2x CLK2 }
CLK2,               {Input Clock = MBUS System Clock }
!RESET,             {Reset, Active Low }
MBB,                {MBUS Busy, Active Low}
MRQ0,               {MBUS Channel 0 Request, Active Low}
MRQ1,               {MBUS Channel 1 Request, Active Low}
MRQ2,               {MBUS Channel 2 Request, Active Low}
MRQ3(node=9),       {MBUS Channel 3 Request, Active Low}
CS,                 {Decoded Processor Chip Select}
WE,                 {Processor Write Enable}
D0,                 {Data Bus Bit 0, Lalched Priority Bit 0}
D1,                 {Data Bus Bit 1, Latched Priority Bit 1}
D2,                 {Data Bus Bit 2, Latched Priority Enable Bit}

                    {OUTPUTS}

!MGT0(node=15),     {MBUS Channel 0 Grant, Active Low}
!MGT1,              {MBUS Channel 1 Grant, Active Low}
!MGT2,              {MBUS Channel 2 Grant, Active Low}
!MGT3,              {MBUS Channel 3 Grant, Active Low}
!EN,                {Settable Priority Enable Bit}
!PRI0(node=23),     {Priority Selection Bit 0}
!PRI1,              {Priority Selection Bit 1}
!CT0,               {Random Counter Bit 0}
!CT1,               {Random Counter Bit 1}
!LP0,               {Latched Priority Bit 0}
!LP1,               {Latched Priority Bit 1}
INT_RST(node=29),   {Sync Reset Node}
ST0(node=31),       {State Variable Bit 0}
ST1,                {State Variable Bit 1}
ST2,                {State Variable Bit 2}
ST3,                {State Variable Bit 3}

                    {End of configuration section}
```

Appendix B. PLD ToolKit Source File for Mbus Arbitrater (continued)

```
EQUATIONS;

INT_RST = RESET;

                    {MBUS Request/Grant Handshake State Machine Equations}

ST3 = <sum> /MRQ3*MRQ1*MRQ0*/PRI1*/ST3*/ST2*/ST0
        +/MRQ3*PRI1*PRI0*/ST3*/ST2*/ST0
        +/MRQ3*MRQ0*/PRI1*/PRI0*/ST3*/ST2*/ST0
        +/MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*/ST0
        +/MRQ2*PRI1*/PRI0*/ST3*/ST2*/ST0
        +MRQ3*/MRQ2*MRQ1*MRQ0*/ST3*/ST2*/ST0
        +MRQ3*/MRQ2*MRQ0*/PRI0*/ST3*/ST2*/ST0
        +MRQ3*/MRQ2*PRI1*/ST3*/ST2*/ST0
        +/MBB*ST3*/ST2*/ST1*/ST0*/MGT3*MGT2*/MGT1*/MGT0
        +/MBB*ST3*/ST2*/ST1*/ST0*MGT3*/MGT2*/MGT1*/MGT0;

ST2 = <sum> MRQ2*/MRQ1*PRI1*/PRI0*/ST3*/ST2*/ST0
        +MRQ2*MRQ1*/MRQ0*/PRI0*/ST3*/ST2*/ST0
        +/MRQ1*/PRI1*PRI0*/ST3*/ST2*/ST0
        +/MRQ0*/PRI1*/PRI0*/ST3*/ST2*/ST0
        +MRQ1*/MRQ0*/PRI1*/ST3*/ST2*/ST0
        +MRQ3*MRQ2*/MRQ1*MRQ0*/ST3*/ST2*/ST0
        +MRQ3*MRQ2*/MRQ1*PRI1*/ST3*/ST2*/ST0
        +MRQ3*MRQ2*MRQ1*/MRQ0*/ST3*/ST2*/ST0
        +/MBB*/ST3*ST2*/ST1*/ST0*/MGT3*/MGT2*/MGT1*MGT0
        +/MBB*/ST3*ST2*/ST1*/ST0*/MGT3*/MGT2*MGT1*/MGT0;

ST1 = <sum> /MBB*/ST3*/ST2*/ST1*ST0*/MGT3*/MGT2*MGT1*/MGT0
        +/MBB*/ST3*/ST2*/ST1*ST0*/MGT3*MGT2*/MGT1*/MGT0
        +/MBB*/ST3*/ST2*/ST1*ST0*MGT3*/MGT2*/MGT1*/MGT0
        +/MBB*/ST3*/ST2*/ST1*ST0*/MGT3*/MGT2*/MGT1*MGT0
        +MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*ST1*/ST0*/MGT3*/MGT2*MGT1*/MGT0
        +MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*ST1*/ST0*/MGT3*MGT2*/MGT1*/MGT0
        +MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*ST1*/ST0*/MGT3*/MGT2*/MGT1*MGT0
        +MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*ST1*/ST0*MGT3*/MGT2*/MGT1*/MGT0;

ST0 = <sum> MBB*/ST3*/ST2*/ST1*ST0*/MGT3*/MGT2*/MGT1*MGT0
        +MBB*/ST3*/ST2*/ST1*ST0*/MGT3*/MGT2*MGT1*/MGT0
        +MBB*/ST3*/ST2*/ST1*ST0*/MGT3*MGT2*/MGT1*/MGT0
        +MBB*/ST3*/ST2*/ST1*ST0*MGT3*/MGT2*/MGT1*/MGT0
        +MBB*/ST3*ST2*/ST1*/ST0*/MGT3*/MGT2*MGT1*/MGT0
        +MBB*ST3*/ST2*/ST1*/ST0*/MGT3*MGT2*/MGT1*/MGT0
        +MBB*/ST3*ST2*/ST1*/ST0*/MGT3*/MGT2*/MGT1*MGT0
        +MBB*ST3*/ST2*/ST1*/ST0*MGT3*/MGT2*/MGT1*/MGT0;

MGT3 = <oe>
       <sum> /MRQ3*MRQ1*MRQ0*/PRI1*/ST3*/ST2*/ST0
        +/MRQ3*PRI1*PRI0*/ST3*/ST2*/ST0
        +/MRQ3*MRQ0*/PRI1*/PRI0*/ST3*/ST2*/ST0
        +/MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*/ST0
        +MBB*/ST3*/ST2*/ST1*ST0*MGT3*/MGT2*/MGT1*/MGT0
        +/MBB*/ST3*/ST2*/ST1*ST0*MGT3*/MGT2*/MGT1*/MGT0
        +MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*ST1*/ST0*MGT3*/MGT2*/MGT1*MGT0
        +MBB*ST3*/ST2*/ST1*/ST0*MGT3*/MGT2*/MGT1*/MGT0
        +/MBB*ST3*/ST2*/ST1*/ST0*MGT3*/MGT2*/MGT1*/MGT0;
```

Appendix B. PLD ToolKit Source File for Mbus Arbitrater (continued)

```
MGT2 = <oe>
       <sum> MBB*/ST3*/ST2*/ST1*ST0*/MGT3*MGT2*/MGT1*/MGT0
         +/MBB*/ST3*/ST2*/ST1*ST0*/MGT3*MGT2*/MGT1*/MGT0
         +/MRQ2*PRI1*/PRI0*/ST3*/ST2*/ST0
         +MRQ3*/MRQ2*MRQ1*MRQ0*/ST3*/ST2*/ST0
         +MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*ST1*/ST0*/MGT3*MGT2*/MGT1*/MGT0
         +MRQ3*/MRQ2*MRQ0*/PRI0*/ST3*/ST2*/ST0
         +MRQ3*/MRQ2*PRI1*/ST3*/ST2*/ST0
         +MBB*ST3*/ST2*/ST1*/ST0*/MGT3*MGT2*/MGT1*/MGT0
         +/MBB*ST3*/ST2*/ST1*/ST0*/MGT3*MGT2*/MGT1*/MGT0;

MGT1 = <oe>
       <sum> MBB*/ST3*/ST2*/ST1*ST0*/MGT3*/MGT2*MGT1*/MGT0
         +/MBB*/ST3*/ST2*/ST1*ST0*/MGT3*/MGT2*MGT1*/MGT0
         +MRQ2*/MRQ1*PRI1*/PRI0*/ST3*/ST2*/ST0
         +/MRQ1*/PRI1*PRI0*/ST3*/ST2*/ST0
         +MRQ3*MRQ2*/MRQ1*MRQ0*/ST3*/ST2*/ST0
         +MRQ3*MRQ2*/MRQ1*PRI1*/ST3*/ST2*/ST0
         +MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*ST1*/ST0*/MGT3*/MGT2*MGT1*/MGT0
         +/MBB*/ST3*ST2*/ST1*/ST0*/MGT3*/MGT2*MGT1*/MGT0
         +MBB*/ST3*ST2*/ST1*/ST0*/MGT3*/MGT2*MGT1*/MGT0;

MGT0 = <oe>
       <sum> MBB*/ST3*/ST2*/ST1*ST0*/MGT3*/MGT2*/MGT1*MGT0
         +/MBB*/ST3*/ST2*/ST1*ST0*/MGT3*/MGT2*/MGT1*MGT0
         +MRQ2*MRQ1*/MRQ0*/PRI0*/ST3*/ST2*/ST0
         +/MRQ0*/PRI1*/PRI0*/ST3*/ST2*/ST0
         +MRQ1*/MRQ0*/PRI1*/ST3*/ST2*/ST0
         +MRQ3*MRQ2*MRQ1*/MRQ0*/ST3*/ST2*/ST0
         +MRQ3*MRQ2*MRQ1*MRQ0*/ST3*/ST2*ST1*/ST0*/MGT3*/MGT2*/MGT1*MGT0
         +/MBB*/ST3*ST2*/ST1*/ST0*/MGT3*/MGT2*/MGT1*MGT0
         +MBB*/ST3*ST2*/ST1*/ST0*/MGT3*/MGT2*/MGT1*MGT0;

              {Random Counter Equations}

CT1 = <oe>
      <sum>        CT1*/CT0
                   +/CT1*CT0;

CT0 = <oe>
      <sum>        /CT0;
```

Appendix B. PLD ToolKit Source File for Mbus Arbitrater (continued)

```
                {Latched Priority Equations}

EN = <oe>
       <sum> /CS*/WE*D2
         +EN*WE
          +EN*CS;

LP1 = <oe>
       <sum> /CS*/WE*D1
         +LP1*WE
          +LP1*CS;

LP0 = <oe>
       <sum> /CS*/WE*D0
         +LP0*WE
          +LP0*CS;

                {Priority Selection Latch}

PRI1 = <oe>
       <sum> /EN*CT1
           +EN*LP1;

PRI0 = <oe>
       <sum> /EN*CT0
         +EN*LP0;

                {End of file}
```

NOTES:

Index

CYPRESS SEMICONDUCTOR

Sales Representatives and Distribution

Direct Sales Offices

California

Cypress Semiconductor Corporate Headquarters
3901 N. First Street
San Jose, CA 95134
Tel: (408) 943-2600
Telex: 821032 CYPRESS SNJ UD
TWX 910 997 0753
FAX: (408) 943-2741

Cypress Semiconductor
23586 Calabasas Rd., Ste 201
Calabasas, CA 91302
Tel: (818) 884-7800
FAX: (818) 348 6307

Cypress Semiconductor
2151 Michelson Dr., Suite 240
Irvine, CA 92715
Tel: (714) 476-8211
FAX: (714) 476-8317

Cypress Semiconductor
16496 Bernardo Center Dr., Suite 215
San Diego, CA 92128
Tel: (619) 487-9446
FAX: (619) 485-9716

Colorado

Cypress Semiconductor
4851 Independence St., Suite 189
Wheat Ridge, CO 80033
Tel: (303) 424-9000
FAX: (303) 424-0627

Florida

Cypress Semiconductor
804 E. Church Street
Orlando, FL 32801
Tel: (407) 422-1890
FAX: (407) 841-9927

Florida

Cypress Semiconductor
10014 N. Dale Mabry Hwy, Suite 101
Tampa, FL 33618
Tel: (813) 968-1504
FAX: (813) 968-8474

Illinois

Cypress Semiconductor
1530 E. Dundee Rd., Suite 190
Palatine, IL 60067
Tel: (312) 934-3144
FAX: (312) 934-7364

Maryland

Cyress Semiconductor
5457 Twin Knolls Rd., Suite 103
Columbia, MD 21045
Tel: (301) 740-2087
FAX: (301) 992-5887

Massachusetts

Cypress Semiconductor
2 Dedham Place, Suite 1
Dedham, MA 02026
Tel: (617) 461-1778
FAX: (617) 461-0607

Minnesota

Cypress Semiconductor
14525 Hwy. 7, Suite 115
Minnetonka, MN 55345
Tel: (612) 935-7747
FAX: (612) 935-6982

New York

Cypress Semiconductor
244 Hooker Ave., Suite B
Poughkeepsie, NY 12603
Tel: (914) 485-6375
FAX: (914) 485-7103

Direct Sales Offices (continued)

New York

Cypress Semiconductor
3 Nob Hill Dr.
Smithtown, NY 11787
Tel: (215) 639-6663
FAX: (516) 544-4359

North Carolina

Cypress Semiconductor
10805 Brass Kettle Road
Raleigh, NC 27614
Tel: (919) 870-0880
FAX: (919) 870-0881

Oregon

Cypress Semiconductor
6950 SW Hampton St., Suite 230
Portland, OR 97223
Tel: (503) 684-1112
FAX: (503) 684-1113

Pennsylvania

Cypress Semiconductor
2 Neshaminy Interplex, Suite 203
Trevose, PA 19047
Tel: (215) 639-6663
FAX: (215) 639-9024

Texas

Cypress Semiconductor
Great Hills Plaza
9600 Great Hills Trail, Suite 150W
Austin, TX 78759
Tel: (512) 338-0204
FAX: (512) 338-0865

Cypress Semiconductor
333 W. Campbell Rd., Suite 220
Richardson, TX 75080
Tel: (214) 437-0496
FAX: (214) 644-4839

Cypress Semiconductor
Twelve Greenway Plaza, Suite 1100
Houston, TX 77046
Tel: (713) 621-8791
FAX: (713) 621-8793

Belgium

Cypress Semiconductor International
51 Rue Du Moulin A Papier, Bte, 11
1160 Brussels, Belgium
Tel: (32) 2 672 2220
Telex: 64677 CYPINT B
FAX: (32) 2 660 0366

France

Cypress Semiconductor France
Miniparc Bat. no 8
61 Avenue des Andes
A.A. de Courtaboeuf
91952 Les Lilis Cedex, France
Tel: (33) 1169-07-5546
FAX: (33) 1169-07-5571

Germany

Cypress Semiconductor GmbH
Hohenlindner Str. 6
D-8016 Feldkirchen, W. Germany
Tel: (49) 089 903 10 71
FAX: (49) 089 903 84 27

Japan

Cypress Semiconductor Japan K.K.
Fuchu-Minami Bldg., 2F
9052-3, 1-Chome, Fuchu-Cho,
Fuchu-Shi, Tokyo, Japan 183
Tel: (81) 423-69-8211
FAX: (81) 423-69-8210

Sweden

Cypress Semiconductor Scandinavia
Kanalvagen 17
18330 Taby, Sweden
Tel: (46) 8 758 2055
FAX: (46) 8 792 1560

United Kingdom

Cypress Semiconductor U.K.
3, Blackhorse Lane,
Hitchin,
Hertfordshire, SG4 9EE
Tel: (44) 0462-420566
FAX: (44) 0462-421969

North American Sales Representatives

Alabama

CSR Electronics
303 Williams Ave., Suite 931
Huntsville, AL 35801
Tel: (205) 533-2444
FAX: (205) 536-4031

Arizona

Thom Luke Sales
2940 N. 67th Place, Suite H
Scottsdale, AZ 85251
Tel: (602) 941-1901
FAX: (602) 941-4127

California

TAARCOM
451 N. Shoreline Blvd.
Mountain View, CA 94043
Tel: (415) 960-1550
FAX: (415) 960-1999

Canada

E.S.P.
5200 Dixie Road, Suite 201
Mississauga, Ontario Canada L4W 1E4
Tel: (416) 626-8221
FAX: (416) 238-3277

E.S.P.
447 McLeod Street, Suite 3
Ottawa, Ontario Canada K1R 5P5
Tel: (613) 236-1221
FAX: (613) 236-7119

E.S.P.
4250 Sere Street
St. Laurent, Quebec H4T 1A6
Tel: (514) 737-9344
FAX: (514) 737-4128

Connecticut

HLM
3 Pembroke Road
Danbury, CT 06813
Tel: (203) 791-1878
FAX: (203) 791-1876

Florida

CM Marketing
1435D Gulf to Bay Blvd.
Clearwater, FL 34615
Tel: (813) 443-6390
FAX: (813) 443-6312

CM Marketing
6091-A Buckeye Ct.
Tamarac, FL 33319
Tel: (305) 722-9369
FAX: (305) 726-4139

CM Marketing
804 E. Church St.
Orlando, FL 32801
Tel: (407) 341-9924
FAX: (407) 841-9927

Georgia

CSR Electronics
1651 Mt. Vernon Rd., Suite 200
Atlanta, GA 30338
Tel: (404) 396-3720
FAX: (404) 394-8387

Illinois

Micro Sales Inc.
54 West Seegers Road
Arlington Hts., IL 60005
Tel: (312) 956-1000
FAX: (312) 956-0189

Indiana

Technology Marketing Corp.
599 Industrial Dr.
Carmel, IN 46032
Tel: (317) 844-8462
FAX: (317) 573-5472

Technology Marketing Corp.
4630-10 W. Jefferson Blvd.
Ft. Wayne, IN 46804-6800
Tel: (219) 432-5553
FAX: (219) 432-5555

Iowa

Midwest Tech. Sales
1930 St. Andrews N.E.
Cedar Rapids, IA 52402
Tel: (319) 393-5115
FAX: (319) 393-4947

North American Sales Representatives (continued)

Kansas

Midwest Tech. Sales
21901 Lavista
Goddard, KS 67052
Tel: (316) 794-8565

Midwest Tech. Sales
15301 W. 87 Parkway, Suite 200
Lenexa, KS 66219
Tel: (913) 888-5100
FAX: (913) 888-1103

Kentucky

Technology Marketing Corp.
4012 Du Pont Circle, Suite 414
Louisville, KY 40207
Tel: (502) 893-1377
FAX: (502) 896-6679

Michigan

Techrep
2550 Packard Road
Ypsilanti, MI 48197
Tel: (313) 572-1950
FAX: (313) 572-0263

Missouri

Midwest Tech. Sales
1314 Robertridge St.
Charles, MO 63303
Tel: (314) 441-1012
FAX: (314) 447-3657

Midwest Tech. Sales
4637 Chippewa Way
St. Charles, MO 63303
Tel: (314) 441-1012
FAX: (314) 447-3657

New Jersey

HLM
1300 Route 46
Parsippany, NJ 07054
Tel: (201) 263-1535
FAX: (201) 263-0914

New Mexico

Quatra Associates
9704 Admiral Dewey N.E.
Albuquerque, NM 87111
Tel: (505) 821-1455
FAX: (602) 820-7054

New York

HLM
64 Mariners Lane
PO Box 328
Northport, NY 11768
Tel: (516) 757-1606
FAX: (516) 757-1636

Reagan/Compar
3215 E. Main Street
Endwell, NY 13760
Tel: (607) 754-2171
FAX: (607) 754-4270

Reagan/Compar
3449 St. Paul Blvd.
Rochester, NY 14617
Tel: (716) 338-3198

Reagan/Compar
41 Woodberry Road
New Hartford, NY 13413
Tel: (315) 732-3775

Ohio

KW Electronic Sales
8514 N. Main Street
Dayton, OH 45415
Tel: (513) 890-2150
FAX: (513) 890-5408

KW Electronic Sales
3645 Warrensville Center Rd., #244
Shaker Heights, OH 44122
Tel: (216) 491-9177
FAX: (216) 491-9102

Pennsylvania

KW Electronic Sales
4485 William Flynn Hwy.
Allison Park, PA 15101
Tel: (412) 487-4300
FAX: (412) 487-4841

L. D. Lowery
2801 West Chester Pike
Broomall, PA 19008
Tel: (215) 356-5300
FAX: (215) 356-8710

North American Sales Representatives (continued)

Puerto Rico

ETS, Inc.
PO Box 10758,
Caparra Heights Station,
San Juan, Puerto Rico 00922
Tel: (809) 798-1300
FAX: (809) 798-3611

Tennessee

CSR Electronics
4314 Woodlawn Pike
Knoxville, TN 37920
Tel: (615) 577-1317
FAX: (615) 577-1306

Texas

Southern States Marketing
400 E. Anderson Lane, Suite 111
Austin, TX 78752
Tel: (512) 835-5822
FAX: (512) 835-1404

South States Marketing
1143 Rockingham, Suite 106
Richardson, TX 75080
Tel: (214) 238-7500
FAX: (214) 231-7662

Utah

Sierra Technical Sales
4700 South 900 East, Suite 3-150
Salt Lake City, UT 84117
Tel: (801) 566-9719
FAX: (801) 565-1150

Washington

Electronic Sources
1603 116th Avenue NE, Suite 115
Bellevue, WA 98004
Tel: (206) 451-3500
FAX: (206) 451-1038

Wisconsin

Micro Sales Inc.
16800 W. Greenfield Ave., Suite 116
Brookfield, WI 53005
Tel: (414) 786-1403
FAX: (414) 786-1813

International Sales Representatives

Australia

Breamac Pty. Ltd.
1045-1047 Victoria Rd.
West Ryde, N.S.W. 2114
Tel: (61) 02-858-5085

Austria

Hitronik Vertriebs GmBH
St. Veitgasse 51
A-1130 Wien, Austria
Tel: (43) 222 824199
FAX: (43) 222 828 55 72

Belgium

Microtronica
Research Park Zellik
Pontbeeklaan 43
B-1730 Asse Zellik
Tel: (32) 02-466-7260
FAX: (32) 02-466-4697

Denmark

A/S Nordisk Electronik
Transformervej 17
DK-2730 Herlev, Denmark
Tel: (45) 02 842000
FAX: (45) 02 921552

Finland

OY Fintronic AB
Italahdenkatu 22
00210 Helsinki, Finland
Tel: (358) 0-6926002
FAX: (358) 0-674886

France

Newtek
8 Rue de L'Esterel, Silic 583
F-94663 Rungis Cedex, France
Tel: (33) 1-4687-2200
FAX: (33) 1 4687 8049

Jermyn + Generim
73/79, Rue des Solets
Silic 585
94663 Rungis Cedex, France
Tel: (33) 1-4978-4400
FAX: (33) 1-4978-0599

France

Jermyn + Generim
31, Domaine Chevalier
83440 Tourrettes, France
Tel: (33) 1-9476-2585

Jermyn + Generim
60 Rue Pierre Cazenueve
31200 Toulouse, France
Tel: (33) 1-6157-9695

Jermyn + Generim
Immeuble Saint-Christophe
Rue de la Frebardiere
B.P. 42-Z.I. Sud Est
35135 Chantepie, France
Tel: (33) 1-9941-7044
FAX: (33) 1-9950-1128

Jermyn + Generim
60 Rue des Acacias
Herrin, France
59147 Gondecourt, France
Tel: (33) 1-2032-3095

Germany

A.P.I. Elektronik GmbH
Lorenz-Braren-Str. 32
D-8062 Markt, Indersdorf, W. Germany
Tel: (49) 8136 7092
FAX: (49) 8136 7398

Astek GmbH
Gottlieb-Daimler-Str.7
D-2358 Kaltenkirchen, W. Germany
Tel: (49) 4191 8711
FAX: (49) 4191 8249

Metronik GmbH
Leonhardsweg 2, Postfach, #1328
D-8025 Unterhaching b. Munich, W. Germany
Tel: (49) 89-611080
FAX: (49) 89 611 6468

Metronik GmbH
Laufamholzstrabe 118
D-8500 Nurnberg, W. Germany
Tel: (49) 9 11 59 00 61

Metronik GmbH
Siemensstrabe 4-6
D-6805 Heddesheim, W. Germany
Tel: (49) 62 03 47 01

International Sales Representatives (continued)

Germany

Metronik GmbH
Semerteichstrabe 92
D-4600 Dortmund 30, W. Germany
Tel: (49) 2 31 42 30 37
FAX: (49) 02 31-41 82 32

Metronik GmbH
Buckhorner Moor 81
D-2000 Norderstedt bei Hamburg, W. Germany
Tel: (49) 040 522 8091
FAX: (49) 040 522 8093

Metronik GmbH
Loewenstrabe 37
D-7000 Stuttgart 70, W. Germany
Tel: (49) 7 11 76 40 33
FAX: (49) 7 11 76 51 81

Metronik GmbH
Gutenbergstr. 5
D-7550 Rastatt, W. Germany
Tel: (49) 7222-69464
FAX: (49) 7222-69896

Hong Kong

Tekcomp Electronics
514 Bank Centre
636 Nathan Road
Kowloon, Hong Kong
Tel: (852) 710-8721
FAX: (852) 3 710 9220

Israel

Talviton Electronics
PO Box 21104, 9 Biltmore St.
Tel Aviv 61 210, Israel
Tel: (972) 3-444572
FAX: (972) 3-455626

Italy

Cramer Italia s.p.a.
134, Via C. Colombo
I-00147 Roma, Italy
Tel: (39) 6-517-981
FAX: (39) 6 514 0722

Dott.Ing.Giuseppe De Mico s.p.a.
V. Le Vittorio Veneto, 8
I-20060 Cassina d'Pecchi
Milano, Italy
Tel: (39) 2 95 20 551
FAX: (39) 2 952 2227

Japan

Tomen Electronics Corp.
2-1-1 Uchisaiwai-Cho, Chiyoda-Ku
Tokyo, Japan 100
Tel: (81) 3 506 3670
FAX: (81) 3 506 3497

C. Itoh Techno-Science Co. Ltd.
8-1, 4-Chome, Tsuchihashi,
Miyamae-Ku, Kawasaki-shi,
Kanagawa, Japan 213
Tel: (81) 44-852-5121
FAX: (81) 44-877-4268

Fuji Electronics Co., Ltd.
Ochanomizu Center Bldg.
3-2-12 Hongo, Bunkyo-ku
Tokyo, Japan 213
Tel: (81) 03-814-1411
FAX: (81) 03-814-1414

International Semiconductor Inc. (ISI)
The Second Precisa Bldg.
4-8-3 Iidabashi Chiyoka-ku
Tokyo, Japan
Tel: (81) 03-264-3301
FAX: (81) 03-264-3419

Korea

Hanaro Corporation
Daeyoung Bldg., 2nd Floor
643-8, Yeoksam-Dong, Kangnam-Ku
Seoul, 150-010 Korea
Tel: (82) 02-558-1144
FAX: (82) 02-558-0157

Netherlands

Semicon B.V.
Gulberg 33
NL-5672 AD Nuenen
Tel: (31) 040 837 075
FAX: (31) 040 838 635

International Sales Representatives (continued)

Norway

Nortec Electronics A/S
Smedsvingen 4, P.O. Box 123
N-1364 Hvalstad, Norway
Tel: (47) 2 846-210
FAX: (47) 2 846-545

Singapore

Desner Electronics
190 Middle Rd., #16-17
Fortune Center, Singapore 0718
Tel: (65) 33 73 180
FAX: (65) 33 73180

Spain

Comelta S.A.
Pedro IV. 8 4-5 Planta
08005 Barcelona, Spain
Tel: (34) 3007-7128

Comelita S.A.
Emilio Munez, 41 Nave 1-1-2
E-Madrid 17
Tel: (34) 1 754 3001
FAX: (34) 1 754 2151

Sweden

Nordisk Electronik AB
P.O. Box 36
Torshamnsgatan 39,
S-164 93 Kista, Sweden
Tel: (46) 8 703 4630
FAX: (46) 8 703 9845

Switzerland

BASIX fur Electronik AG
Forrlibuckstrasse 150, Postrach
CH-8010 Zurich, Switzerland
Tel: (41) 01-276-1100
FAX: (41) 01-820-3441

Taiwan R.O.C.

Prospect Technology Corp.
5, Lane 55, Long-Chiang Road
Taipei, Taiwan
Tel: (886) 2 721 9533
FAX: (886) 2 773 3756

United Kingdom

Pronto Electronic System LTD.
City Gate House
399-425 Eastern Ave.
Gants Hill Ilford, Essex IG2 6LR, U.K.
Tel: (44) 1 554 62 22
FAX: (44) 1 518 32 22

Ambar Cascom Ltd.
Rabans Close
Aylesbury Bucks HP19 3R5, U.K.
Tel: (44) 0296 434 141
FAX: (44) 0296 296 70

Distribution

Anthem Electronics

Arizona
Tempe, AZ 85281
(602) 966-6600

California
Irvine, CA 92718-2809
(714) 768-4444

Rocklin, CA 95834
(916) 624-9744

San Jose, CA 95131
(408) 453-1200

San Diego, CA 92121
(619) 453-9005

Chatsworth, CA 91311
(818) 775-1333

Colorado
Englewood, CO 80112
(303) 790-4500

Connecticut
Meriden, CT 06450
(203) 237-2282

Florida
Clearwater, FL 34623
(813) 796-2900

Georgia
Norcross, GA 30093
(404) 381-0768

Illinois
Elk Grove, IL 60007
(312) 640-6066

Massachusetts
Wilmington, MA 01887
(503) 657-5170

Maryland
Columbia, MD 21045
(301) 995-6640

Minnesota
Eden Prairie, MN 55344
(612) 944-5454

New Jersey
Fairfield, NJ 07006
(201) 227-7960

New York
Hauppauge, NY 11787
(516) 273-1660

Ohio
Worthington, OH 43085
(614) 888-9707

Oregon
Beaverton, OR 97005
(503) 643-1114

Pennsylvania
Horsham, PA 19044
(215) 443-5150

Texas
Richardson, TX 75081
(214) 238-7100

Utah
Salt Lake City, UT 84119
(801) 973-8555

Washington
Bothel, WA 98052
(206) 483-1700

Distribution

Arrow Electronics

Alabama
Huntsville, AL 35816
(205) 837-6955

Arizona
Phoenix, AZ 85040
(602)437-0750

California
Chatsworth, CA 91311
(818) 701-7500

San Diego, CA 92123
(619) 565-4800

Sunnyvale, CA 94089
(408) 745-6600

Tustin, CA 92680
(714) 669-4524

Canada
Montreal, Quebec, Canada H4P1W1
(514) 735-5511

Mississauga, Ontario, Canada L5T1H3
(416) 670-7769

Neapean, Ontario, Canada K2E7W5
(613) 229-6903

Quebec, Que., Canada G1N2C9
(418) 871-7500

Colorado
Englewood, CO 80112
(303) 790-4444

Connecticut
Wallingford, CT 06492
(203) 265-7741

Florida
Deerfield Beach, FL 33441
(305) 429-8200

Lake Mary, FL 32746
(407) 333-9300

Georgia
Norcross, GA 30071
(404) 449-8252

Illinois
Itasca, IL 60143
(312) 250-0500

Indiana
Indianapolis, IN 46241
(317) 243-9353

Kansas
Lenexa, KS 66214
(913) 541-9542

Maryland
Columbia, MD 21046
(301) 995-6002

Massachusetts
Wilmington, MA 01887
(617) 658-0900

Michigan
Ann Arbor, MI 48108
(313) 971-8220

Grand Rapids, MI 49508
(616) 243-0912

Minnesota
Edina, MN 55435
(612) 830-1800

Distribution

Arrow Electronics (continued)

New Hampshire
Manchester, NH 03013
(603) 668-6968

New Mexico
Albuquerque, NM 87106
(505) 243-4566

New Jersey
Parsippany, NJ 07054
(201) 538-0900

New York
Hauppauge, NY 11788
(516) 231-1000

Marlton, NY 08053
(609) 596-8000

Corporate Headquarters
Melville, NY 11747
(516) 391-1300

Rochester, NY 14623
(716) 427-0300

North Carolina
Raleigh, NC 27604
(919) 876-3132

Ohio
Centerville, OH 45459
(513) 435-5563

Solon, OH 44139
(216) 248-3990

Oklahoma
Tulsa, OK 74146
(918) 252-7537

Oregon
Beaverton, OR 97006
(503) 645-6456

Pennsylvania
Monroeville, PA 15146
(412) 856-7000

Texas
Austin, TX 78758
(512) 835-4180

Carrollton, TX 75006
(214) 380-6464

Houston, TX 77099
(713) 530-4700

Washington
Kent, WA 98032
(206) 575-4420

Wisconsin
Brookfield, WI 53005
(414) 792-0150

Distribution

Marshall Industries

Alabama
Huntsville, AL 35801
(205) 881-9235

Arizona
Phoenix, AZ 85044
(602) 496-0290

California
Corporate Headquarters
El Monte, CA 91731-3004
(818) 459-5500

Chatsworth, CA 91311
(818) 407-0101

Irvine, CA 92718
(714) 458-5355

San Diego, CA 92131
(619) 578-9600

Milpitas, CA 95035
(408) 942-4600

Rancho Cordova, CA 95670
(916) 635-9700

Canada
Montreal, Ontario
(514) 848-9112

Ottawa, Ontario
(613) 564-0166

Colorado
Thornton, CO 80241
(303) 451-8383

Connecticut
Wallingford, CT 06492-0200
(203) 265-3822

Florida
Ft. Lauderdale, FL 33309
(305) 977-4880

Altamonte Springs, FL 32701
(305) 767-8585

St. Petersburg, FL 33702
(813) 576-1399

Georgia
Norcross, GA 30093-9990
(404) 923-5750

Illinois
Schaumburg, IL 60195
(312) 490-0155

Indiana
Indianapolis, IN 46278
(317) 297-0483

Kansas
Lenexa, KS 66214
(913) 492-3121

Maryland
Silver Springs, MD 20910
(301) 622-1118

Massachusetts
Wilmington, MA 01887
(508) 657-9029

Michigan
Livonia, MI 48150
(313) 525-5850

Minnesota
Plymouth, MN 55441
(612) 559-2211

Distribution

Marshall Industries (continued)

Missouri
Bridgetown, MO 63044
(314) 291-4650

New Jersey
Fairfield, NJ 07006
(201) 882-0320

Mt. Laurel, NJ 08054
(609) 234-9100

New York
Johnson City, NY 13790
(607) 798-1611

Hauppauge, NY 11788
(516) 273-2424

Rochester, NY 14624
(716) 235-7620

North Carolina
Raleigh, NC 27604
(919) 878-9882

Ohio
Dayton, OH 45424
(513) 898-4480
Solon, OH 44139
(216) 248-1788

Oregon
Beaverton, OR 97005
(503) 644-5050

Pennsylvania
Pittsburgh, PA 15238
(414) 963-0441

Texas
Austin, TX 78754
(512) 837-1991

Carrollton, TX 75006
(214) 770-0604

Houston, TX 77040
(713) 795-9200

Utah
Salt Lake City, UT 84115
(801) 485-1551

Washington
Bothell, WA 98011
(206) 486-5747

Wisconsin
Waukesha, WI 53186
(414) 797-8400

Falcon Electronics

Connecticut
Milford, CT 06460
(203) 878-5272

Massachusetts
Franklin, MA 01701
(508) 520-0323

New York
Hauppauge, NY 11788
(516) 724-0980

Distribution

Semad

Canada

Toronto

Markham, Ontario L3R 4Z4
(416) 475-3922
FAX: (416) 475-4158

Montreal

Point Claire, Quebec H9R 427
(514) 694-0860
1-800-363-6610

Ottawa

Ottawa, Ontario K2C 0R3
(613) 727-8325
FAX: (613) 727-9489

Vancouver

Burnaby, B.C. V3N 4S9
(604) 420-9889
1-800-663-8956
FAX: (604) 420-0124

Calgary

Calgary, Alberta T2H 2S8
(403) 252-5664
FAX: (403) 255-0966

Zeus Components Inc.

California

Agoura Hills, CA 91301
(818) 889-3838

Yorba Linda, CA 92686
(714) 921-9000

San Jose, CA 95131
(408) 998-5121

Florida

Oviedo, FL 32765
(305) 365-3000

Massachusetts

Lexington, MA 02173
(617) 863-8800

Maryland

Columbia, MD 21045
(301) 997-1118

Zeus Components Inc. (continued)

New York

Port Chester, NY 10573
(914) 937-7400

Ronkonkoma, NY 11779
(516) 737-4500

Ohio

Dayton, OH 45439
(513) 293-6162

Texas

Richardson, TX 75081
(214) 783-7010

Cypress Semiconductor
3901 North First Street
San Jose, CA 95134
(408) 943-2600