VESA®

2150 North First Street, Suite 440
San Jose, CA. 95131-2029

Phone: (408) 435-0333
FAX: (408) 435-8225

VESA VL-Bus
Plug and Play Addendum
Preliminary

Version: 0.1
Revision 5
Revision Date: April 11, 1994

Purpose

To provide a standard mechanism to allow VL bus cards to participate in existing plug and play standards.

Summary

This addendum describes the method to allow VL bus cards to participate in the ISA bus plug and play standard. The method described here adheres to the VL bus goal of software transparency, by allowing the ISA plug and play algorithms and software to remain unchanged

This addendum requires a working knowledge of the ISA Plug and Play specification, as only the specifics to the VL bus are described.

Intellectual Property

Trademarks

VESA, VL-Bus **Video Electronics Standards Association**

All other trademarks mentioned in this document are property of their respective owners.

Patents

This and other VESA documents are adopted by the Video Electronics Standards Association as standards without regard as to whether their adoption may involve patents on articles, materials, or processes. Such adoption does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standards documents.

Revision History

0.1 R4	**Preliminary copy**
0.1 R5	**Updated ISA plug and play register reference to reflect changed Plug and Play ISA spec.**

1. Introduction

1.1. Background of the VL-Bus™ Standard

Some of the goals of the VL bus were to be software transparent for devices moved from the ISA bus to VL, to be easy to implement, and to be low cost on the motherboard. These goals have contributed heavily to the rapid success of the VL bus.

It also prevented the inclusion of plug and play in the VL bus, which would have involved system changes. In reality, this has not been much of a problem, since the devices which are connected to the VL bus generally have a well established set of IO and interrupts. Since each VL slot is required to have own set of -LREQ/-LGNT signals, there is no resource contention for these signals, as there is for ISA DMA channels. The most popular cards are:

VGA. **The addresses are fixed and do not conflict. VGAs have an interrupt, always on IRQ2 (actually IRQ9 redirected to 2) which is virtually never used and is provided only for compatibility with some older test programs.**

IDE. **The address and interrupt are fixed. The only issue is selecting the port for multiple IDEs, but the VL IDE will likely be at the primary address. Also, the secondary IDE may be implemented on the VL card. Conflicts between VL and ISA IDE controllers cure themselves since the VL cycle prevents an ISA cycle.**

SCSI **Generally a master. No issue with DMA channels as there is with ISA. The IO address and interrupt are the only issues. SCSI IO addresses generally do not collide with other devices. Since a BIOS is required, which has a customary address (C800), the interrupt can generally be selected as one of the higher ones, where there are less contentions.**

LAN **The address and interrupt must be selected. Most VL bus LAN controllers are probably masters.**

1.2. Background of ISA Plug and Play

Since the adoption of the VL bus, a standard has emerged for plug and play for ISA cards. This standard does not require any modification to the motherboard, and allows non-plug and play cards to coexist with newer cards incorporating the plug and play hardware. Software supporting this standard is expected to begin appearing in BIOSes and operating systems.

1.3. Related Documents

The information in this application note assumes knowledge of both the VL bus and the ISA plug and play algorithms. The following specifications should be referred to:

VESA VL-Bus 2.0 November 1, 1993

Plug and Play ISA Specification Version 1.0 May 28, 1993

2. Plug and Play for VL

Defining a separate plug and play for VL would require special VL software, which would be one more thing BIOS vendors would have to write and support. It would result in some systems having plug and play for ISA, but not VL, and vice-versa. It would also get away from the VL goal of software transparency.

The proposed VL plug and play has the following features:

- **Uses ISA bus Plug and Play software**
- **Transparent to the Plug and Play software.**
- **VL devices use the ISA plug and play hardware algorithm with some added details.**
- **Works on existing VL motherboards without modification.**

2.1. Plug and Play register access

The ISA plug and play uses three registers: Address Register (write only), Write Data (write only), and Read Data (read only). Accesses to these registers must be shared by all devices, both on the VL bus and the ISA bus. The write only registers are easily handled. VL devices must not capture the cycles with LDEV# or return ready. They must allow the writes to pass onto the ISA bus. The VL devices capture the data as it is written. There is one exception to this, where the VL device does claim a write cycle, which is discussed below.

Read cycles are more complicated. For most reads, only one Plug and Play device is selected for reads at any given time. If a VL device is selected, it will claim the read cycle by driving LDEV# and returning LRDY# like a normal IO read. If it is not selected, it will not drive LDEV#.

The Isolation cycles perform IO reads from the data port in which any plug and play device may drive the data bus and all plug and play devices must monitor the data bus during reads. Since the VL and ISA cards are connected to different data busses this is not possible. During the ISA access the VL data bus will be driven by the system logic. If the VL cards pull LDEV# the ISA devices will not see the access.

The solution is as follows. Assume that there are 2 VL and 2 ISA plug and play cards for this example:

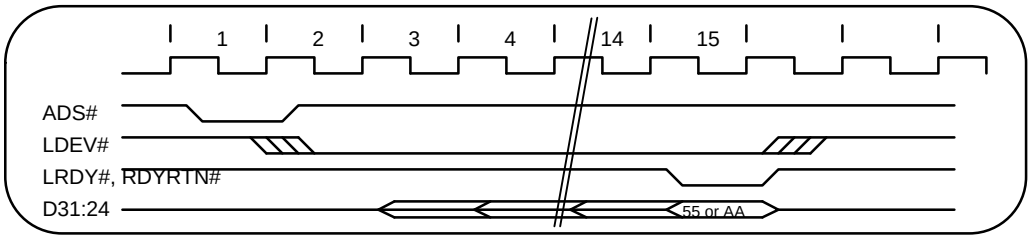
A. At the start of the VL isolation process the VL devices drive LDEV# and return LRDY# for the isolation reads. The ISA cards will not see the cycles, and will do nothing. The software will find a VL card, and isolate it. The VL card must pull LDEV# and drive LRDY# when its Card Select Number register is written to prevent the ISA cards from seeing the cycle (since the ISA cards did not see the isolation register reads, none of them will have gone to the "sleep" state).

B. On the next round, the card which was already isolated does not participate, so the second VL card is isolated.

C. On the third and forth rounds, both VL cards have been isolated, so neither drives LDEV#. The reads go to the ISA bus and an ISA cards are found.

This is transparent to the plug and play software and ISA cards.

Multiple VL cards will be capturing the isolation reads and driving LRDY# simultaneously. This is the only case where multiple devices will be driving LRDY# at one time. Care must be taken to avoid bus contentions. All VL cards must drive LRDY# low on the same clock edge. A number of wait states must be picked that all cards will use for these cycles. Thirteen wait states (a 15 T-state access) has been picked, which will allow enough time for the card to do any data shifting, etc.



VL bus plug and play rules (over and above Plug and Play spec).

A VL bus card in the ISOLATION state should drive LDEV# and LRDY# for:

Reads from the Read Data Port which access the Serial Isolation Register (01)

Writes to the Write Data Port when the Card Select Number is targeted (reg 06).

A VL bus card in the CONFIG state should drive LDEV# and LRDY# for:

Reads from the Read Data Port regardless of the register number (in the config state only one card is selected).

The data bus may be driven only when LDEV# is pulled low by the card.

On Isolation reads:

Must always be thirteen wait states (15 T states total) when captured by a VL card.

When the card drives the data (current ID bit = 1):

- **The data bus must be driven from Hi Z to the proper level directly to avoid bus contentions.**
- **D23:0 are driven with 0s, D31:24 are driven with 55 or AA as per the ISA Plug and Play spec.**
- **Data may be driven starting on the any T state after the first T2.**

When the card does not drive the data (current ID bit = 0):

- **The data bus must not be driven for any part of the cycle.**
- **The data (bits 25 and 24) is monitored at the end of the final (fifteenth) T state to determine whether any other card is driving it (as per the ISA Plug and Play spec).**
- **The card still drives LDEV# and LRDY# (if all VL cards have a 0 in the ID bit position the cycle must still be claimed on the VL bus).**

Plug and Play isolation sequence

	Event	VL Cards	ISA Cards
1	**Power Up**	**Enter WAIT FOR KEY state**	**Enter WAIT FOR KEY state**
2	**CPU sends Initiation Key**	**Enter Sleep mode**	**Enter Sleep mode**
3	**Wake(0) command**	**Enter Isolation state**	**Enter Isolation State**
4	**Serial Isolation Register Reads**	**Drive -LDEV and -LRDY. Drive data as per spec.**	**Do not see these cycles**
5	**After isolation reads:**	**One VL card remains in isolation state, all others in sleep**	**All ISA cards still in isolation state.**
6	**Set CSN**	**Card in isolation state drives -LDEV and -LRDY, capturing the cycle. Cards in Sleep state ignore**	**Do not see these cycles.**
7	**Wake(0) command**	**Cards which have not yet been found (CSN not programmed yet) enter the isolation state.**	**All ISA cards enter isolation state (they are already there).**
8	**Steps 4 to 7 repeat for each VL card**		
9	**Serial Isolation Register Reads**	**All VL cards are now ignoring**	**ISA card respond to these cycles in the normal manner.**
10	**After isolation reads:**		**One ISA card remains.**
11	**Set CSN**		**Remaining card gets its CSN programmed.**
12	**Wake (0) command**		**remaining cards go back to isolation state.**
13	**Steps 9 to 12 repeat for each ISA card**		
14	**Final iteration finds no cards and the algorithm terminates.**		
15	**remaining accesses**	**VL devices allow writes to go the ISA bus. They capture (by driving -LDEV and returning -RDY) reads when they are selected.**	

Assumptions:

This scheme assumes that the ISA cards will respond properly to the following sequence

Wake=0 command
Write of 01 to the Address Port (point to serial isolation register)
(72 reads of serial isolation register are NOT seen by the ISA card)
Write of 06 to the Address Port (point to CSN register)
(write to data port NOT seen by the ISA card)

The above sequence repeats until all VL cards are found, then normal ISA plug and play sequences follow.

2.2. Plug and Play Resource Data and configuration registers

The Plug and Play ISA Specification defines a Resource Data format in which the device indicates what its IRQ, DMA, IO and Memory requirements are. It also defines configuration registers for the Plug and Play software to set the resources to be used by the card. The 1.0 version of the Plug and Play ISA spec supports only the 16Mbytes of memory address space which is available on the ISA bus. 32 bit memory descriptors and configuration registers are in proposed additions to the spec.

Rather than extend the current 24 bit descriptors, a new descriptor type is being added for 32 bit address ranges. A 32 bit Fixed Location Range Descriptor is also being added for memory ranges which cannot be moved.

Likewise a new 32 bit memory configuration space is being added for the Plug and Play software to assign a 32 bit address.

VL cards may use any of the descriptor types in the ISA plug and play spec. The DMA resources would normally only be used if the card also contained an ISA peripheral.

If a VL card uses the original memory range descriptor and configuration registers, (which support 24 bit addressing) it must decode the upper bits to 0s. It will also ignore the Memory Control bits, which sets the data width (VL targets are fixed at 32 or 16 bit for the transfer size, and must always drive all 32 data bit on the bus).

Below is a brief overview of the added descriptors and registers. Consult the Plug and Play ISA Spec for more details.

IMPORTANT: The following information is preliminary and is under the control of the Plug and Play ISA Committee. Please refer to the Plug and Play ISA Specification before implementation.

32 Bit Memory Range Descriptor

Size	Field Name	Description
byte	**Memory Range Descriptor**	**Value = 1000 0101 B (Type=1, Large item name=5**
byte	**Length, byte [07:00]**	**Value = 0001 0001 B (17 byte descriptor)**
byte	**Length, byte [15:08]**	**Value = 0000 0000 B**
byte	**Information**	**Same definition as Item 1 memory descriptor EXCEPT:** **Bits 4:3** **11 = 32 bit memory only.**
byte	**Range min base address [07:00]**	**Address bits [07:00] of the minimum base memory address for which the card may be configured.**
byte	**Range min base address [15:08]**	**As above, bits [15:08]**
byte	**Range min base address [23:16]**	**As above, bits [23:16]**
byte	**Range min base address [31:24]**	**As above, bits [31:24]**
byte	**Range max base address [07:00]**	**Address bits [07:00] of the maximum base memory address for which the card may be configured.**
byte	**Range max base address [15:08]**	**As above, bits [15:08]**
byte	**Range max base address [23:16]**	**As above, bits [23:16]**
byte	**Range max base address [31:24]**	**As above, bits [31:24]**
byte	**Base alignment [07:00]**	**This field contains bits [07:00] of the base alignment. The base alignment provides the increment for the minimum base address.**
byte	**Base alignment [15:08]**	**As above, bits [15:08]**
byte	**Base alignment [23:16]**	**As above, bits [23:16]**
byte	**Base alignment [31:24]**	**As above, bits [31:24]**
byte	**Range Length bits [07:00]**	**This field contains bits [07:00] of the memory range length. The range length provides the length of the memory range in 1 byte blocks.**
byte	**Range Length bits [15:08]**	**As above, bits [15:08]**
byte	**Range Length bits [23:16]**	**As above, bits [23:16]**
byte	**Range Length bits [31:24]**	**As above, bits [31:24]**

32 Bit Fixed Location Memory Range Descriptor

Size	Field Name	Description
byte	**Memory Range Descriptor**	**Value = 1000 0110 B (Type=1, Large item name=6**
byte	**Length, byte [07:00]**	**Value = 0001 1001 B (9 byte descriptor)**
byte	**Length, byte [15:08]**	**Value = 0000 0000 B**
byte	**Information**	**Same definition as Item 1 memory descriptor EXCEPT:** **Bits 4:3** **11 = 32 bit memory only.**
byte	**Range base address [07:00]**	**Address bits [07:00] of the base memory address for which the card may be configured.**
byte	**Range base address [15:08]**	**As above, bits [15:08]**
byte	**Range base address [23:16]**	**As above, bits [23:16]**
byte	**Range base address [31:24]**	**As above, bits [31:24]**
byte	**Range Length bits [07:00]**	**This field contains bits [07:00] of the memory range length. The range length provides the length of the memory range in 1 byte blocks.**
byte	**Range Length bits [15:08]**	**As above, bits [15:08]**
byte	**Range Length bits [23:16]**	**As above, bits [23:16]**
byte	**Range Length bits [31:24]**	**As above, bits [31:24]**

32 Bit Memory Space Configuration

Name	Register	Description
Base Address [31:24] descriptor 0	**0x76**	**Read/write value indicating the selected memory base address bits [31:24] for 32 bit memory descriptor 0.**
Base Address [23:16] descriptor 0	**0x77**	**As above, bits [23:16]**
Base Address [15:08] descriptor 0	**0x78**	**As above, bits [15:08]**
Base Address [07:00] descriptor 0	**0x79**	**As above, bits [07:00]**
32 bit memory control	**0x7A**	**32 bit Memory Control** **Bit 0 Read only** **0=lower registers are the range length** **1=lower registers are the upper limit** **Bits 2:1 Read/write.** **00 = 8 bit memory** **01 = 16 bit memory** **10 = (Reserved)** **11 = 32 bit memory** **Bits 7:3 (Reserved)**
Memory Limit or upper range [31:24]	**0x7B**	**Read/write value indicating the selected high address bits or range length for 32 bit memory descriptor 0 bits [31:24]**
Memory Limit or upper range [23:16]	**0x7C**	**As above, bits [23:16]**
Memory Limit or upper range [15:08]	**0x7D**	**As above, bits [15:08]**
Memory Limit or upper range [07:00]	**0x7E**	**As above, bits [07:00]**
Filler	**0x7F**	**Reserved**
32 bit memory descriptor 1	**0x80-0x88**	**32 Bit Memory descriptor 1**
Filler	**0x89-0x8F**	**Reserved**
32 bit memory descriptor 2	**0x90-0x98**	**32 Bit Memory descriptor 2**
Filler	**0x99-0x9F**	**Reserved**
32 bit memory descriptor 3	**0xA0-0xA8**	**32 Bit Memory descriptor 3**