**Configuration Address Space Format**• PCI configuration space is divided into a separate, dedicated configuration address space for each function contained within a PCI device (i.e.. in a chip or on a card). • Host Bridge Needn't Implement Configuration Space

The first 16 dwords of a function's configurations pace is referred to  as the function's configuration Header space. The format and usage  of this area are defined by the specification. Three Header formats  are currently defied:  • Header Type Zero for all devices other than PCI‐to‐PCI bridges. Header Type One for PCI‐to‐PCI bridges. • Header Type Two for CardBus bridges (defined in the CardBus spec).

**Configuration Mechanism**

This mechanism utilizes two 32‐bit IO ports : • The 32‐bit Configuration Address Port, occupying IO addresses 0CF8h ~ 0CFBh. • The 32‐bit Configuration Data Port, occupying IO addresses 0CFCh ~ 0CFFh.

Accessing one of a PCI function's configuration registers is a two step process: • STEP 1. Write the target bus number, device number, function number and dword number to the Configuration Address Port and set the Enable bit in it to one. • STEP 2. Perform a one‐byte, two‐byte, or four‐byte IO read from or an write to the Configuration Data Port.

• In response, the host/PCI bridge compares the specified target bus to the range of buses that exist on the other side of the bridge and, if the target bus resides beyond the bridge, it initiates a PCI configuration read or write (based on whether the processor is performing an IO read or write with the Configuration Data Port).

**Configuration address port**

The assertion of reset clears the port to all zeros. Any 8‐ or 16‐bit access within this IO dword is passed directly onto the PCI bus as an 8‐ or 16‐bit PCI IO access. The 32‐bits of information written to the Configuration Addres s Port must conform to the template shown on the next slide: • bits [1:0] are hard‐wired, read‐only and must return zeros when read. • bit 31 must be set to a one, enabling the translation of a subsequent processor IO access to the Configuration Data Port into a configuration access on the PCI bus. • If bit 31 is zero and the processor initiates an IO read from or IO write to the Configuration Data Port, the transaction is passed through to the PCI bus as a PCI IO transaction.

**Configuration data port**

-Bus Compare, and Data port Usage

• a Bus Number register (in a chipset that only supports one host/PCI bridge, the bridge may have a bus number register that is hardwired to 0, a read/write register that reset forces to 0, or it Just implicitly knows that it is the bridge to PCI bus 0) • and a Subordinate Bus Number register.

If bit 31 in the Configuration Address Port is enabled (set to one), the bridge compares the target bus number to the range of buses that exists beyond the bridge. • If the target bus is the same as the value in the Bus Number register, this is a request to perform a configuration transaction on PCI bus 0. A subsequent IO read from or write to the bridge's Configuration Data Port at 0CFCh causes the bridge to generate a Type 0 configuration read or write transaction. • When devices that reside on a PCI bus detect a Type 0 configuration transaction in progress, this informs them that one of them is the target device. • If the target bus specified In the Configuration Address Port does not match the value in the bridge's Bus Number register, but is equal to or less than the value in the bridge's Subordinate Bus Number register, the bridge converts the subsequent processor IO access to its Configuration Data Port into a Type 1 configuration transaction on its PCI bus. • When devices that reside on a PCI bus (other than PCI‐to‐PCI bridges) detect a Type 1 configuration access in progress, they ignore the transaction.

The only devices on a PCI bus that pay attention to the Type 1 configuration transaction are PCI‐to‐PCI bridges. • If the target bus is not within range, then a PCI‐to‐PCI bridge ignores the Type 1 access. • If it's in range, the access is passed through the PCI‐to‐PCI bridge as either a Type 0 configuration transaction (if the target bus compares to the bridge's Secondary Bus Number register), or as a Type 1 transaction (the target bus number is equal to or less than the value in the bridge's Subordinate Bus Number register).

A screenshot of a cell phone

Description automatically generated

**Type 0 Configuration Transaction vs type 1 configuration transaction**

**Type 0 configuration**

**Address Phase** • During any PCI transaction, all PCI devices on the bus latch the following information at the end of the address phase: • The contents of the AD bus. ϖIn a configuration transaction, this consists of the target function, Configuration dword and 00b on the least‐significant two bits if it's a Type 0 configuration transaction, or the target bus number, device number, function number, dword number and 01b on the least‐ significant two bits if it's a Type 1 configuration transaction. • The state of the FRAME# signal (asserted, indicating that a valid address and command have been latched from the bus). • The state of the IDSEL# signal (only has meaning if this is a Type 0 configuration transaction). ϖThe PCI device (i.e., package) that samples its IDSEL asserted is the target PCI device. The bridge implements a separate IDSEL signal for each PCI device implemented on its secondary bus

During any PCI transaction, all PCI devices on the bus latch the following information at the end of the address phase: • The command on C/BE#[3:0]. In this case, it indicates that this is a configuration read or write transaction. The command type is derived from the type of access the processor is performing with the host/PCI bridge's Configuration Data Port. An IO read converts to a configuration read and an IO write converts to a configuration write. • 00b on AD[1:0] indicates that this is a Type 0 configuration transaction targeting one of the devices on this PCI bus. In essence, the 00b is a shorthand way of indicating that the bridge has already done the target bus comparison and established that the transaction is targeting a device on this bus.

• The PCI device that samples its IDSEL asserted is the target device. • AD[1:0] are 00b, ‘Type 0’ transaction targeting one of the devices on this bus. • AD[7:2] : the target configuration dword. • AD[10:8]: the target function within the physical device. • AD[31:11]: reserved

• The target device number specified in bits [15:11] in the Configuration Address Port are decoded within the bridge) and the decoder asserts the appropriate IDSEL output signal during the transaction's address phase. • If the bridge's device decoder determines that there is no device (or card slot) implemented at the target device position on its secondary bus, it will not assert any IDSEL outputs during the transaction's address phase. The transaction will end up experiencing a Master Abort because no target will assert DEVSEL# to claim the transaction.

• Device Number field within the Configuration Address Port: a 5‐bit field (device number from 0‐ through‐31d as the target device). Obviously, a PCI bus with 32 devices implemented on it (each one presenting a load to the bus) is not going to function correctly. • In reality, a 33MHz PCI bus typically supports no more than 10 devices on the bus. • The bottom line is that most of the 32d possible device positions on any PCI bus are going to be unoccupied. • There is no rule. however, that dictates which device position each device must occupy. • They don't even have to be implemented as contiguous device number, but that wouldn't make a lot of sense.

A screenshot of a cell phone

Description automatically generated

**Type 1 Configuration Access Example**

• When any PCI‐to‐PCI bridge latches a Type 1 configuration access (command = configuration read or write and AD[I:0] = 01 b) on its primary side, it must determine which of the following actions to take: • ACTION 1. If the bus number field on the AD bus doesn't match the number of its secondary bus and isn't within the range of its subordinate buses, the bridge should ignore the access. • ACTION 2. If the bus number field matches the bus number of its secondary bus, it should claim and pass the configuration access onto its secondary bus as a Type 0 configuration access. AD[1:0] on the secondary bus are set to 00 b (Indicating a Type 0 access). AD[10:2] (target function and dword) are passed as is to its secondary AD bus. The device number field is decoded within the bridge to select one of the IDSEL lines to assert on the secondary bus. The configuration command is passed from the primary to the secondary C/BE bus.

• ACTION 3. If the bus number field isn't equal to its secondary bus, but is within the range of buses that are subordinate to the bridge, the bridge claims and passes the access through as a Type 1 access. AD[31:0] (target bus, device, function and dword) are passed to the secondary AD bus as is. AD[1:0] are set to 01b, indicating that a Type 1 access is in progress on the secondary bus. The configuration command is passed from the primary to the secondary C/BE bus. • Each PCI‐to‐PCI bridge is required to implement a Primary Bus Number register, a Secondary Bus Number register, and a Subordinate Bus Number register. • PCI‐to‐PCI bridges on PCI bus 0 are discovered and numbered during the configuration process.

**Implementation of IDSEL**

The IDSEL outputs can be Implemented in one of two ways by the host/PCI bridg e designer. • Be aware that PCI‐to‐PCI bridges must use the first method. **Method One ‐ lDSELs Routed Over Unused AD Lines** • This is the method used by most host/PCI bridge and system board designers. • The upper twenty‐one address lines AD[31:11], are not used during the address phase of a Type 0 configuration access. The system board designer is therefore free to use these signal Lines as IDSEL signals to the various physical PCI packages (up to twenty‐one of them). • Internally, the bridge decodes the target Device number contained in Configuration Address Port bits [15:11] to select which AD line to set to one. Each of these address lines is connected to the IDSEL input of a separate PCI device.

• This approach places an additional load on the AD line, however, and it's a rule that each PCI device is only permitted to place one electrical load on each PCI bus signal.

• As an example of upper AD line‐to‐IDSEL mapping, AD11 could be set to one if the target device is device zero (the first PCI device); AD12 set to on e If the target device is device one, etc. The specification suggests (it's only a suggestion) the following mapping. The bridge internally decodes the device number field in the Configuration Address Port and selects an IDSEL signal to assert. Rather than implementing IDSEL output pins, the IDSEL signals internal to the bridge are directed to AD[31:16] in the following manner: θ The IDSEL associated with device 0 is connected to AD16. θ The lDSEL associated with device i is connected to AD17. θ The IDSEL associated with device 2 is connected to AD18, etc. θ The IDSEL associated with device 15 is connected to AD31. θ For devices 16 through 31, none of the upper AD lines should be asserted when the Type 0 configuration transaction is performed. Since no device detects its IDSEL asserted, none respond, resulting in a Master Abort by the bridge.

• This approach supports the implementation of 16 devices on a PCI bus (from a configuration standpoint; from a realistic electrical loading standpoint, you're only going to place 10 to 11 loads on a 33MHz bus and have it work correctly).

**Method Two IDSEL Output Pin/Traces** The host/PCI bridge designer can decode bits [15:11] (target physical Device number) in the Configuration Address Port and assert the target physical device's IDSEL output signal line. This method requires the implementation of a separate IDSEL output pin on the bridge for each physical package on the PCI bus and a separate point‐to‐point IDSEL trace on the system board between the bridge and each physical PCI device or connector. Most host/PCI bridge designs don't implement IDSEL output pins because it is a pin‐ and trace‐intensive solution.

**PCI-to-PCI Bridge Configuration Registers vs PCI Function Configuration Registers**

**lntro to Configuration Header Region** • Each PCI function possesses a block of 64 configuration dwords reserved for the implementation of its configuration registers. The format and usage of the first 16 dwords is predefined by the PCI specification. This area is referred to as the device's Configuration Header Region (or Header Space). The specification currently defines three Header formats, referred to as Header types Zero, One and Two. • Header Type One is defined for PCI‐to‐PCI bridges. A full description of Header Type One can be found in "Configuration Registers" on page 552. • Header Type Two is defined for PCI‐to‐CardBus bridges and is fully defined in the PC Card spec. • Header Type Zero is used for all devices other than PCI‐to‐PCI and cardBus bridges. This chapter defines Header me Zero.

**Registers Used to Identify Device's Driver**The OS uses some combination of the following mandatory registers to determine which driver to load for a device: • Vendor ID. • Device ID. • Revision.  • Classcode. • SubSystem Vendor ID. • Subsystem ID.

**Vendor ID Register** • The value FFFFh is reserved and must be returned by the host/PCI bridge when an attempt is made to perform a configuration read from a non‐existent device's configuration register. The read attempt results in a Master Abort. θ The Master Abort is not considered to be an error θ The bridge must none the less set its Received Master Abort bit in its configuration Status register

**Cache Line Size Register** • This read/write configuration register specifies the system cache Line size in dword increments (e.g., a P6‐based system would store the value O8h, indicating a cache line size of eight dwords, or 32 bytes). • The register must be implemented by bus masters that implement the Memory Write‐and‐Invalidate command. Because it must know the cache line size in order to ensure that it starts transactions on a cache line boundary and keeps its promise to write an entire line into memory. • The bus master may not use the Memory Write‐and‐Invalidate command when this register is set to zeros (which indicates that the configuration software hasn't yet told it the cache line size). In this case, the master should only use Memory Write transactions to update memory. • A device may limit the number of cache line sizes that it supports. If an unsupported value is written to the register by the configuration software, the device behaves as if the value zero was written.

BIST Register • Optional. This register may be implemented by both master and  target devices.

**Base Address Registers (BARS)** The Base Address Registers (BARS) located in dwords 4‐through‐9 of the device's configuration Header space, are used to implement a function's programmable memory and/or IO decoders. • Each register is 32‐bits wide (or 64‐bits wide if it's a memory decoder and its associated memory block can be located above the 4GB address boundary) • Required if a device implements memory and/or IO decoders. • Bit 0 = 0, the register is a memory address decoder. • Bit 0 = 1, the register is an IO address decoder.

Memory‐Mapping Recommended • In a PC environment, IO space is densely populated and will only become more so in the future. • The specification strongly recommends that the device designer provide only a Memory Base Address Register that maps a device's register set into memory space. Optionally, an IO Base Address Register may also be included to map it into IO space, but this is not recommended. • This gives the configuration software the flexibility to map the device's register set into memory space and, if an IO Base Address Register is also provided, into IO space as well.

A screenshot of a cell phone

Description automatically generated

• Base Address Field: This field consists of bits [31:4] for a 32‐bit memory decoder and bits [63:4] for a 64‐bit memory decoder.

A screenshot of a cell phone

Description automatically generated

**Interrupt Pin Register** • The read‐only Interrupt Pin register defines which of the four PCI interrupt request pins, INTA#‐through‐INTD#, a PCI function is connected to.

**lnterrupt Line Register** • The Interrupt Line register is used to identify which input on the interrupt controller the function's PCI interrupt request pin (as specified in its Interrupt Pin register) is routed to. • For example, in a PC environment the values 00h‐through‐0Fh in this register (refer correspond to the IRQ0‐through‐IRQ15 inputs on the interrupt controller.

**Min‐Gnt Register: Timeslice Request** • Optional for a bus master and not applicable to non‐master devices. • The value indicates how long a burst period the device needs (in increments of 1/4 of a microsecond, or 250 us). • The Max‐Lat register and the Min‐Gnt register and are registers used  by the configuration software to determine: • how often a bus master typically requires access to the PCI bus and • the duration of a typical transfer when it does acquire the bus.

**Max‐Lat Register: Priority‐Level Request** •Optional for a bus master and not applicable to non‐master devices. • The specification states that this read‐only register specifies "how  often" the device needs access to the PCI bus (in increments of 1/4 of  a microsecond, or 250us).  •value of zero indicates the device has no stringent requirement in  this area. • The Max\_Lat register value indicates how often the master would like  to have access to the bus (i.e., receive its GNT# from the arbiter

**Primary PCI Bus Number Register, Secondary PCI Bus Number Register and Subordinate PCI Bus Number Register**

**Primary Bus Number register.** Initialized by software with the number of the bridge's upstream PC1 bus. The host/PCI bridge is only connected to one PC1 bus, so it only implements a Bus Number register. The host/PCI bridge doesn't have to implement a Secondary Bus Number register (because it is irrelevant

**Secondary Bus Number register.** Initialized by software with the number of the bridge's downstream PC1 bus.

**Subordinate Bus Number register.** Initialized by software with the highest numbered PC1 bus that exists on the secondary side. If the only bus on the bridge's downstream side is the bridge's secondary bus, then the Secondary and Subordinate Bus Number registers would be initialized with the num- ber of the secondary bus.

**Basic Transaction Filtering Mechanism**

• The configuration program assigns all IO devices that reside behind a PCI‐to‐PCI bridge mutually‐exclusive address ranges that are blocked together within a common overall range of IO locations. • The PCI‐to‐PCI bridge passes any IO transactions detected on the primary side of the bridge to the secondary side if the target address is within the range associated with the community of IO devices that reside behind the bridge.

• Any IO transactions detected on the secondary side of the bridge are passed to the primary side if the target address is outside the range associated with the community of IO devices that reside on the secondary side (because the target device doesn't reside on the secondary side, but may reside on the primary side).

• All memory‐mapped IO devices (i.e.. non‐prefetchable memory) that reside behind a PCI‐to‐PCI bridge are assigned mutually‐exclusive memory address ranges within a common block of memory locations.

• All memory devices (i.e., regular memory, not memory‐mapped IO) that reside behind a PCI‐to‐PCI bridge are assigned mutually‐exclusive memor y address ranges within a common overall range of memory locations. The PCI‐to‐PCI bridge is then programmed to pass any memory transactions detected on the primary side of the bridge to the secondary side if the target address is within the range associated with the community of memory devices that reside behind the bridge. • Conversely, any memory transactions detected on the secondary side of the bridge are passed to the primary side if the target address is outside the range associated with the community of memory devices that reside on the secondary side (because the target device doesn't reside on the secondary side, but may reside on the primary side).

A picture containing text, map

Description automatically generatedA close up of a map

Description automatically generatedA picture containing text, map

Description automatically generatedA close up of a map

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated