Cache (handout)

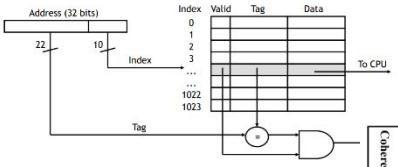
Direct Mapped Cache

A direct-mapped cache is the simplest approach: each main memory address maps to exactly one cache block.

-taking advantage of spatial locality, we read 4 bytes at a time.

Advantages: simple, fast

Disadvantage: Mapping is fixed



Set Associative Cache

♣ An intermediate possibility is a set-associative cache. — The cache is divided into groups of blocks, called sets.

— Each memory address maps to exactly one set in the cache, but data may be placed in any block within that set.

N-way set associative: N entries for each Cache index.

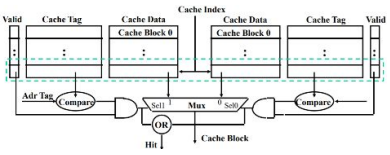
-N direct mapped caches operates in parallel

EX: two-way set associative cache

-Cache index selects a "set" from the cache

-The two tags in the set are compared to the input in parallel

-Data is selected based on the tag result



Disadvantages of set Associative cache

-N-way set associative cache: N-way comparators vs 1, extra mux delay for the data, Data comes after Hit/miss decision and set selection

- in a direct mapped cache, Cache Block is available before hit/mis:

Fully Associative Cache

Permits data to be stored in any cache block, instead of forcing each memory address into one particular block.

— When data is fetched from memory, it can be placed in any unused block of the cache.

— This way we'll never have a conflict between two or more memory addresses which map to a single cache block.

-forget about the cache index,

-compare the cache tags of all cache entries in parallel

Ex) block size = 32 B blocks, we need N 27-bit comparators

By definition miss = 0 for a fully associative cache

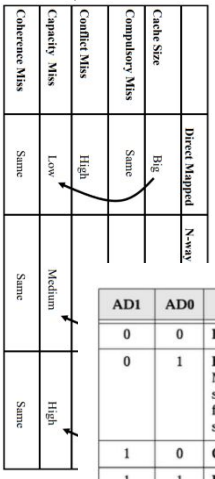
Disadvantages:

However, a fully associative cache is expensive to implement. — Because there is no index field in the address anymore, the entire address must be used as the tag, increasing the total cache size. — Data could be anywhere in the cache, so we must check the tag of every cache block. That's a lot of comparators!

Addressing sequence

Linear (Sequential Mode)

All memory devices that support multiple data phase transfers must support linear addressing. When the Memory Write-and-Invalidate command is used. the start address must be aligned on a cache line boundary and it must indicate (on AD[I:O]) linear addressing. At the completion of each data phase, (even if the initiator isn't asserting any Byte Enables in the next data phase) the memory target increments its address counter by four to point to the next sequential dword for the next data phase (or the next sequential quadword if performing 64-bit transfers)



Design Options at

Table 10-1: Memory Burst Address Sequence

AD1	AD0	Addressing Sequence
0	0	**Linear**, or sequential, addressing sequence during the burst.
0	1	**Reserved**. Prior to revision 2.1, this indicated Intel Toggle Mode addressing. When detected, the memory target should signal a Disconnect with data transfer during the first data phase or a disconnect without data transfer in the second data phase.
1	0	**Cache Line Wrap** mode. First defined in revision 2.1.
1	1	**Reserved**. When detected, the memory target should signal a Disconnect with data transfer during the first data phase or a disconnect without data transfer in the second data phase.

Cache Line Wrap Mode

Support for Cache Line Wrap Mode is optional and is only used for memory reads. A memory target that supports this mode must implement the Cache Line Size configuration register (so it knows when the end of a line has been reached).

•

The start address can be any dword within a line. At the start of each data phase of the burst read, the memory target increments the dword address in its address counter. Implementation of Cache Line Wrap Mode is optional for memory and meaningless for IO and configuration targets.

PCI IO Addressing

•During an IO transaction, the start IO address placed on the AD bus during the address phase has the following format: •AD[31:2] identify the target dword of IO space. •

AD[1:0] identify the least-significant byte (i.e.. the start byte) within the target dword that the indicator wishes to perform a transfer with •

At the end of the address phase, all IO targets latch the start address and the IO read or write command and begin the address decode.

Reflected Wave switching

The PC1 bus is unterminated and uses wavefront reflection to advantage. A carefully selected, relatively weak output driver is used to drive the signal he partially towards the desired logic state (as illustrated at point A in Figure 3-2 on page 27). The driver only has to drive the signal line partially towards its final state, rather than completely (as a strong incident-wave driver would). No inputs along the trace will sample the signal until the next rising-edge of the clock.

When the wavefmnt arrives at the unterminated end of the bus, it is reflected back and doubled (see point B in Figure 3-2 on page 27). Upon passing each device input again during the wavefront's return trip down the trace, a valid logic level registers at the input on each device. The signal is not sampled, however, until the next risingedge of the PC1 clock (point C in the figure). Finally, the wavefront is absorbed by the low-impedance within the driver. This method cuts driver size and surge current in half. There are three timing parameters associated with PC1 signal timing: TVAL, Tprop, Tsu

PCI Clock and Reset Specifications

Symbol	Parameter	Min	Max	Units
T_{cyc}	**CLK** Cycle Time	30	∞	ns
T_{high}	**CLK** High Time	11		ns
T_{low}	**CLK** Low Time	11		ns
-	**CLK** Slew Rate	1	4	V/ns
-	**RST#** Slew Rate	50	-	mV/ns

Figure 3-2: High-Going Signal Reflects and Is Doubled

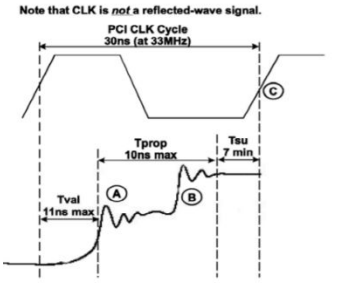
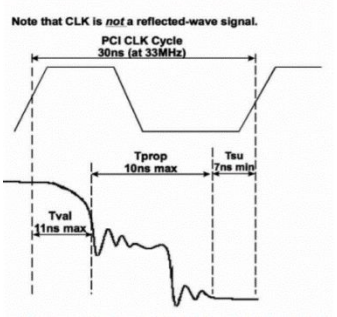


Figure 3-3: Low-Going Signal Reflects and Is Doubled



Cache Handout

What is the average memory access time for the following memory system?

- Level 1 cache: 91% hit rate, 1-cycle access time.
- Level 2 cache: 98% hit rate, 15-cycle access time.
- Memory: 140-cycle access time.

Cache hits and misses both require some number of cycles to access the cache (the access time). You may assume that all memory accesses are hits in main memory.

The main thing to remember is that you incur the access time for a cache regardless of whether it's a hit or a miss. So if we have an access that is not in cache at all, we first spend 1 cycle accessing L1 to find out that it's a miss. Then we spend 15 more cycles accessing L2 to find out that we've missed again, and finally 140 cycles accessing memory.

We have 3 cases…
L1 hit (0.91 probability): 1 cycle
L1 miss, L2 hit (0.09 * 0.98 probability): 1 cycle + 15 cycles
L1/L2 miss, memory hit (0.09 * 0.02 probability): 1 cycle + 15 cycles + 140 cycles

Altogether:
AMAT = 0.91 * 1 + 0.09 * (0.98*(1+15) + 0.02*(1+15+140) = 2.602 cycles

No individual memory access will take 2.602 cycles, but this should be the average over all memory accesses.

Question 2)

A byte-addressable machine with 32-bit memory addresses has a cache with the following properties:

- 16-byte cache blocks
- 8KB of data in the cache
- Direct-mapped
- Write-through

(a) How many cache blocks are there?

cache blocks = (bytes in cache)/(bytes per block) = 8KB/16B = $2^{13}/2^4 = 2^9$ = 512

(b) How many cache sets are there?

In a direct-mapped cache, the number of sets is the same thing as the number of blocks. (Remember, # sets = # blocks/associativity, and here the associativity is 1). So 512.

(c) How many bits of **meta**data are required for each cache entry? Explain what each is for.

We need tag bits and a valid bit. We don't need a dirty bit, since the cache is write-through and we update memory immediately when we get a write.

tag bits = address bits - index bits - offset bits

index bits = enough bits to choose a cache set. Since there are 512 = 2^9 sets, we need 9 index bits to select among them.

offset bits = enough bits to choose a byte within a cache block. Since there are 16 bytes/block, we need 4 offset bits to select among them.

tag bits = 32 - 9 - 4 = 19.

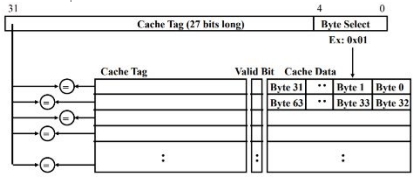
Total metadata bits/block = 19 tag bits + 1 valid bit = 20

Question 3)

Question 5)

Explain the difference between *direct-mapped*, *set-associative*, and *fully associative* cache designs.

In a direct-mapped cache a memory address maps to only one cache block.

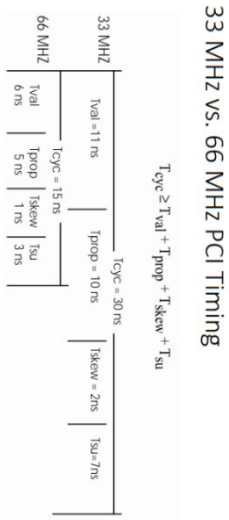


In a fully-associative cache a memory address maps to any cache block. In a set-associative cache a memory address maps to a particular set of cache blocks.
Question 6)
Show how 32-bit addresses are divided into tag, index, and offset given the following cache descriptions:
32KB, byte addressable, 8-way set associative cache with 4 byte blocks.

$32KB = 2^{15}$ bytes.
$\text{Lines} = \frac{2^{15}}{4*8} = 1024$. Thus, 10 index bits.
$\text{Tag} = 32 - (\text{index} + \text{offset}) = 20$.

2 : offset bits
10 : index bits
20 : tag bits

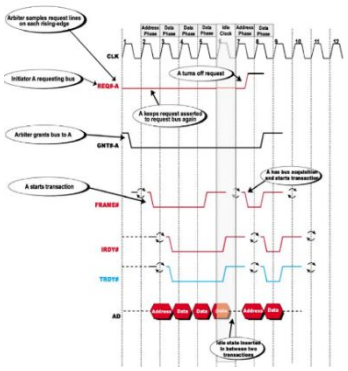
33MHz vs 66 MHz timing



back to back Transaction

•If a bus master desires another access, it should continue to assert its REQ# after it has asserted FRAME# for the First transaction.
•If the arbiter continues to assert GNT# to the master at the end of the first transaction, the master may then immediately initiate a second transaction.
•However, a bus master attempting to perform two, back-to-back transactions usually must insert an idle cycle between the two transactions.
•The designer of a bus master should make an decision as to whether or not it's worth the additional logic it would take to implement the fast back-to-back.

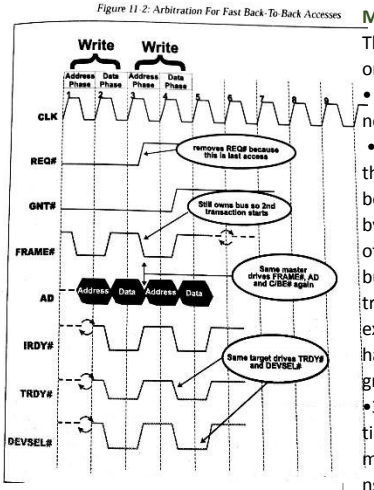
Figure 11-1: Back-to-Back Transactions With an Idle State In-Between



Fast back to back Transactions

•If all of the required criteria are met, the master can eliminate the idle cycle between the two bus transactions. These are referred to as Fast Back-to-Back transactions.
•This can only occur if there is a guarantee that there will not be contention (on any signal lines) between the masters and/or targets involved in the two transactions.
•There are two scenarios where this is the case. 1.
Scenario 1 : Master Guarantees Lack of Contention 2.
Scenario 2: Targets Guarantee Lack of Contention.

Figure 11-2: Arbitration For Fast Back-To-Back Accesses



Advantages of address/data stepping

•Advantages:
Diminished Current Drain and Crosstalk
DATA Stepping: The data presented by the initiator during each data phase of a write transaction is qualified by the assertion of the IRDY# signal by the initiator. The data presented by the target during each data phase of a read transaction is qualified by the assertion of the TRDY# signal by the target. In other words, data can be stepped onto the bus, as well as address.

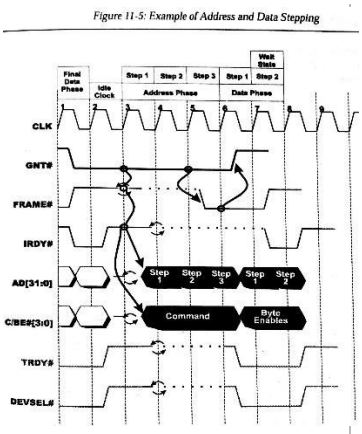
Disadvantages of address/data stepping

•Due to the prolonged period it takes to set up the address or data on the bus, there is a performance penalty associated in any address or data phase where stepping is used.
•In the midst of stepping the address on to the bus, the arbiter may remove the grant from the stepping master.

Stepping implementation

Configuration command register stepping control bit:
CASE 1: if the device is not capable of stepping, the bit is hardwired to zero
CASE 2: if the device always using stepping, the bit is hardwired to one.
CASE3: if the device's ability to use stepping can be enabled and disabled via software, the bit is implemented as a read/writable bit. And reset sets it to one(i.e stepping is enabled).

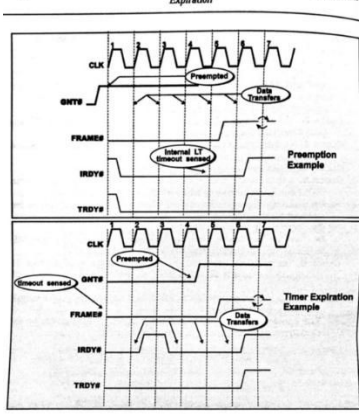
Figure 11-5: Example of Address and Data Stepping



Master initiates termination

The initiator terminates a transaction for one of four reasons:
• 1. The transaction has completed normally.
• 2. The initiator has already used up its the slice and has been living on borrowed time and is then preempted by the arbiter (because one or more other bus masters are requesting the bus) before it completes its burst transfer. • The initiator's Latency Timer expired some time ago and the arbiter has now removed the initiator's bus grant signal (GNT#).
•3. The Initiator is preempted during its time slice and then uses up its allotted time slice before completing its overall transfer.
•4. The initiator has aborted the transaction because no target has responded to the address. This is referred to as a Master Abort.
Preemption During Time slice => preemption example
Timeslice expiration followed by preemption => timer example

Figure 12-1: Master-Initiated Termination Due to Preemption and Master Latency Timer Expiration



Master abort on single vs. multiple-data phase transactions

Two possible cases: •
CASE 1. The initiator starts a single data phase transaction and aborts it due to DEVSEL# not detected. •
CASE 2. The initiator starts a multi-data phase transaction and aborts it due to DEVSEL# not detected.

Figure : Example of Master Abort on Single-Data Phase Transaction (note: this is not a special cycle)

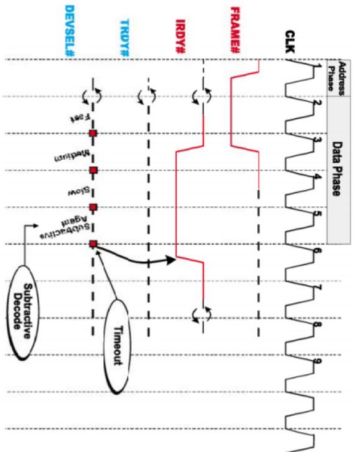
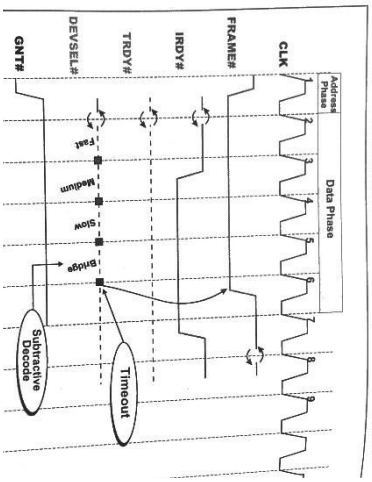


Figure 12-3: Example of Master Abort on Multiple Data Phase Transaction



Action taken by master in response to master abort

General

•Set the Received Master Abort bit in its configuration Status register •Report the error back to the device driver (typically, via an interrupt request).
Master Abort on Special Cycle Transaction •The Received Master Abort status bit should not be set.
•No target is expected to assert DEVSEL# during a Special Cycle and it would be a protocol violation if one did.
Master Abort on Configuration Access • When a Master Abort occurs on a configuration read, the host/PCI bridge must return all ones to the processor.
•On a configuration write, the processor write is permitted to terminate normally (i.e.. as if the data were successfully written).
•This is considered to be an error and the master must set the Received Master Abort bit in its Status register.

Target Initiated termination

A target may use STOP# to tell the initiator to end the transaction on the current data phase.
Using DEVSEL# and TRDY# in conjunction with STOP#, the target can indicate one of four different termination cases to the initiator.
1. **Disconnect With Data Transfer:** •
The target is ready and willing to transfer the current dword, but instructs the master to Disconnect upon completion of

the current data phase (current dword is transferred). This is referred to as a Disc onnect with Data Transfer. •

There are two variants: **Disconnect A an d B**.

a). The initiator may continue the transf er (starting with next dword) at a later ti me,

b). Or the initiator may choose not to re sume the transaction

2. Disconnect Without Data Transfer: • The target refuses to transfer the curren t dword and instructs the master to disc onnect from it without transferring the dword. This is referred to as a Disconnec t Without Data Transfer. •

There are two variants: **Disconnect 1 an d Disconnect 2 Without Data transfer**.

1). The initiator may choose to continue the transfer (starting with the current d word) at a later time

2). Or the initiator may choose not to res ume the transaction (this might occur if i t were prefetching and did not really ne ed the data).

3. Retry:

Issue a Retry during the first data phase. No data is transferred during the transa ction and the initiator is obliged to Retry the transaction later.

4. Target Abort:

Target Abort the transaction and do not Retry (no data is transferred).

Disconnect with data transfer-disconnect A and B

• The target indicates that it wants to transfer the current data item and then dis-connect by asserting TRDY# and STOP# while keeping DEVSEL# asserted. This tells the initiator that the target is ready to transfer the current data item (TRDY# asserted) and wishes to stop the transfer (STOP# asserted). • DEVSEL# remaining asserted gives the initiator permission to resume the transaction later (if so desired: resumption is not mandatary) at the point where it was disconnected (on the dword that would have been transferred in the next data phase). • Disconnect A and B differ from each other by the state of IRDY# at the point where the Disconnect is signaled.

Figure 12-4: Disconnect A With Data Transfer—IRDY# Still Deasserted When Disconnect Issued

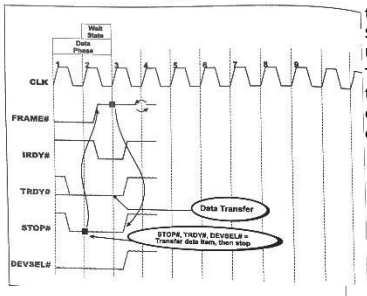
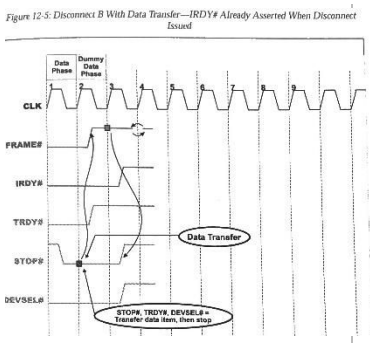


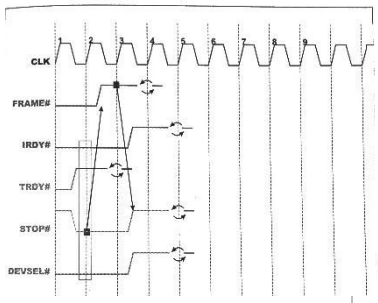
Figure 12-5: Disconnect B With Data Transfer—IRDY# Already Asserted When Disconnect Issued



Disconnect without data transfer – disconnect 1 and 2

Assume that at least one data phase has completed (on the rlsing-edge of clock one) and at least one dword has therefrom been transferred. When the initiator completed the previous data phase and entered the current one (In clock one), it kept IRDY# asserted to indicate that it is ready to complete the cur- Rnt data phase. It did not however, deassert FRAME# because this Isn't the final data phase. In this data phase, the target asserts STOP# (during clock one) and deasserts TRDYi, Indicating that It warn the initiator to terminate the transaction on this data phase with no data transferred. In response, the initiator keeps IRDY# asserted (in clock two) and deasserts FRAME#. indicating that the final data phase is in progress. No data is transferred in this last "dummy" data phase, however, bemuse the target has TRDY# deasserted. In dock three, the initiator deasserts IRDYt, returning the bus to the idle state. The target deas- serts SMP# and DEVSELb.

Figure 12-6: Disconnect 1—When Target Asserts STOP# and Deasserts TRDY#, IRDY# Is Already Asserted

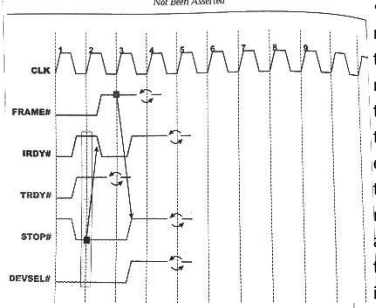


Disconnect 2

Fire 12-7 on page 189 also illustrates a Disconnect Without Data Transfer, but the Initiator deasserted IRDYX upon entry to the current data phase (indicating that it isn't ready to transfer the current data item). When the initiator discovers (on the rising-edge of clock two) that the target wants it to stop on this data phase without transfer- ring the current dword, it responds by asserting IRDY# and deasserting FRAME#. This signals the final "dummy" data phase. The initiator deasserts IRDYX one clock later to return the bus to the idle state and the target deasserts STOP# and DEVSELA.

Upon receipt of a Disconnect Without Data Transfer, the initiator has the option of resuming the transaction later or not. If it was prefetching data that wasn't explicitly required, it may choose not to.

Figure 12-7: Disconnect 2—When Target Asserts STOP# and Deasserts TRDY#, IRDY# Has Not Been Asserted



Retry

• A target only issues Retry to the initiato r if it cannot permit any data to be transf erred. A target only issues Retry to the initiator if it cannot permit any data to be transferred during the current transaction. In other words, it is a rule that if a target is going to issue a Retry to the initiator, it must do it in the first data phase. The signaling for a Retry is identical as that for a Disconnect without Data Transfer except that it occur in the first data phase. Unlike the Disconnect Without Data Transfer, however. the initiator is required to m-attempt the transaction at a later time (while resumption of the transaction is optional if disconnected).

• Retry is indicated to the initiator (by the target) by keeping TRDY# deasserted while asserting STOP# and DEVSEL#. This tells the initiator that the target does not intend to transfer the current (the first) data item (TRDY# deasserted) and that the initiator must stop the transaction on this data phase (STOP# asserted). The transaction must be retried periodically and must be retried identically each time.

Figure 12-8: Retry Received with IRDY# Asserted

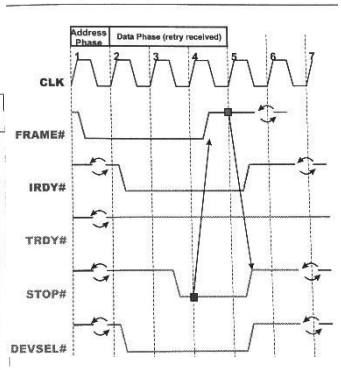
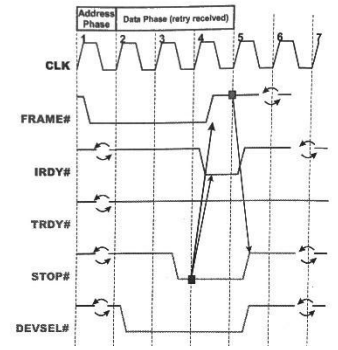


Figure 12-9: Retry Received without IRDY# Asserted



Target Abort

• If a target detect a fatal error or will never be able to respond to the transaction for some other reason, it may signal a Target Abort. It can occur in the first or any other data phase of the transaction. This instructs the initiator to end the transaction and indicates that the target does not want the transaction resumed. • It also means that any data already transferred during this transaction may be corrupted. The initiator must set the Received Target Abort bit in its configuration Status register and the target must set the Signaled Target Abort bit in its configuration Status register

• Target Abort is indicated to the initiator by simultaneously asserting STOP# and deasserting TRDY# and DEVSEL#. This tells the initiator that the target will not transfer the current data item (TRDY# deasserted) and that the initiator must stop the transaction on the current data phase (STOP# asserted). The early deassertion of DEVSEL# instructs the initiator not to re-attempt the transaction and differentiates Target Abort from Disconnect Without Data Transfer.

Figure 12-10: Target Abort Example

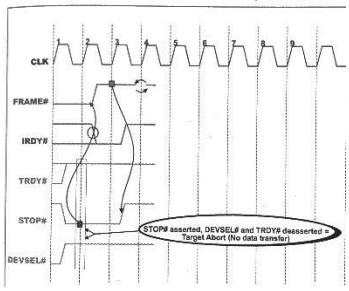


Table 12-1: Target-Initiated Termination Summary

Termination Type	TRDY#	STOP#	DEVSEL#	Data Transfer?	Comments
Disconnect With Data Transfer	asserted	asserted	asserted	some data transferred, including cur- rent dword	If master wants to trans- fer more data, transac- tion will be re-initiated at next dword address.
Disconnect With- out Data Transfer	deasserted	asserted	asserted	some data transferred, but not current dword	Occurs in a data phase other than the first. The master may resume the transaction or not. If resumed, it must start on the same dword.
Retry	deasserted	asserted	deasserted	no	Occurs in first data phase. Master is obliged to Retry the transaction identically
Target Abort	deasserted	asserted	deasserted	perhaps	Fatal error; no Retry.

STOP# not permitted during turn-around cycle

STOP# MUST NOT BE ASSERTED DURING THE TURN AROUND CYCLE THAT IMMEDIAEIY FQLLOWS THE ADDRESS PHASE OF A READ TRANSACTION.

Resumption of disconnected transaction FOR DISCONNECT

Resumption of Disconnected Transactio n is Optional Unlike a Retry, upon receipt of a Disconnect the master may or may not choose to rearbitrate for the bus and continue the transaction at the point of disconnection. As an example, a bridge might continue a memory read transaction past the first data phase to fill up a read-ahead buffer (i.e. a prefetch buffer) in case the originating master on the other side of the bridge wanted to read additional information. If the memory target that it's reading from disconnects from it at some point, the master would probably choose not to resume the transaction at the point of disconnection.

Reasons Target Issues disconnect

Target Slow to Complete Subsequent Data Phase. Assume that the target determines that the latency to complete a data phase (

except the first. which must adhere to the 16 clock rule) will be longer than eight PCI c locks. There are two cases: **CASE 1.** The target determines that it can transfer the current data item within eight clocks and also knows that the master intends to perform another data phase (the master kept FRAME# asserted when it asserted IRDY#). The target has determined that it will not be able to transfer the next data item within eight clocks after entering the next data phase. The target t must assert TRDY# and STOP#, thereby forcing the master to disconnect from it upon completing the transfer of the current data item.

• **CASE 2.** In this case, the target enters a data phase before determining that it cannot transfer a data item within eight clocks. The target must keep TRDY# deasserted and assert STOP# as soon as it determines that it cannot meet the eight clock rule. This forces the master to disconnect from the target without transferring the current data item. • It should be noted that this rule was added in revision 2.1 of the specification. This rule ensures that a slow target will not tie up the bus for extended periods of time.

Reasons target issues retry

• Target Very Slow to Complete First Data Phase. If the latency to first data phase completion will be longer than 16 PCI clocks, the target must issue a Retry to the initiator. The only exception to the 16 dock rule is granted to the host/PCI bridge. It is permitted 32 docks to complete the first data phase. • The 16 clock target first data phase latency rule prevents a target with a very long latency to first data phase completion from monopolizing the bus.

Reasons target issues target Abort

Broken Target. If a target is broken and unable to transfer data, it may indicate this by issuing a Target Abort to the master. • I/O Addressing Error. The byte enable combination is not one supported by the target (in other words, it doesn't "own" all of the addressed locations within the current dword). • Address Phase Parity Error. If a target appears to be addressed in a transaction but there is a address phase parity error, the target may end the transaction by issuing a Target Abort to the master (as well as asserting SERR#).

Master Abort on Other Side of PCI-to-PCI Bridge. When a PCI-to-PCI bridge passes a transaction through the bridge for an initiator and the transaction is not claimed by a target on another bus: • The bridge experiences a Master Abort on the destination bus and sets the Received Master Abort bit in the status register that is associated with th e destination bus (a bridge has two status registers; one for each side). • The bridge then issues a Target Abort to the originating master and sets the Signaled Target Abort bit in its status register that is associated with th e originating bus. • The originating master sets the Received **Target** Abort bit in its Status register and generates an interrupt to have its driver check its status.

Master's response to target abort

In response to a Target Abort, the initiator takes one of the following actions • Generates an interrupt to alert its related device driver to check its status. • Generates SERR# (assuming the master's SERR# Enable bit is set to one in its PCI configuration Command register).

Data parity Generation during read

During each data phase of a read transaction, the target drives data onto the AD bus. It is therefore the target's responsibility to supply correct parity to the initiator on the PAR signal starting one clock after the assertion of TRDY#. At the conclusion of each data phase. it is the initiator's responsibility to latch the contents of AD[31:0] and C/BE#[3:0] and to calculate the expected parity during the clock cycle immediately following the conclusion of the data phase. The initiator then latches the parity bit supplied by the target from the PAR signal on the next rising-edge of the clock and compares computed vs. actual parity. If a miscompare occurs, the initiator then asserts PERR# during the next clock (if it's enabled to do so by a one in the Parity Error Response bit in its Command register). The assertion of PERR# lags the conclusion of each data phase by two PCI clock cycles. The platform design (in other words, the chipset) may or may not include logic that monitors PERR# during a read and takes some system-specific action.

Figure 13-1: Parity on Read Transaction

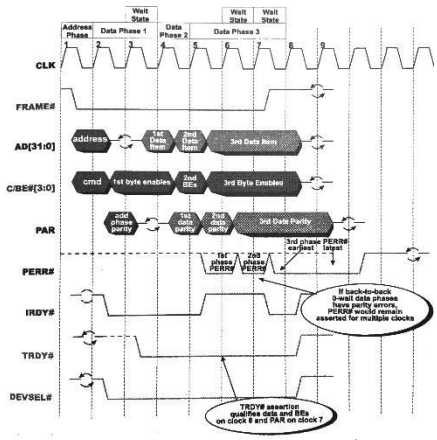
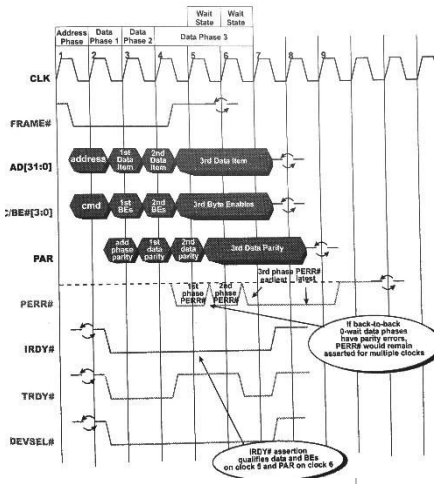


Figure 13-2: Parity on Write Transaction



Data parity Generation during write

During each data phase of a write transaction, the initiator drives data onto the AD bus. It is therefore the initiator's responsibility to supply correct parity to the target on PAR during the clock immediately following the assertion of IRDY#. At the conclusion of each data phase. it is the target's responsibility to latch the contents of AD[31:0] and C/BE#[J:O] and to calculate the expected par- ity during the clock cycle immediately following the conclusion of the data. phase. The target then Latches the PAR bit supplied by the Initiator on the next rising-edge of the clock and compares the computed parity to the actual parity. If a mis compare occurs, the target then asserts PERR# to the master during the next clock. The assertion of PEW lags the conclusion of each data phase by one PC1 dock cycle. During a burst write, it is the initiator's responsibility to sample the state of the PERRt signal on the second rising-edge of the PC1 clock after the conclusion of each data phase. If it samples PERM asserted by the tar- get, this indicates that the last data item written to the target was corrupted in flight "Data Parity Reporting" on page 209 provides a detailed discussion of the actions taken by a bus master upon detection of a data phase parity error.

Data Parity Reporting

• Upon detection of a data phase parity error, the device that checked the parity is responsible for asserting the Detected Parity Error bit in its PCI configuration Status register. It also asserts PERR# if the Parity Error Response bit in its PCI configuration Command register is set to one. Only two categories of devices are excluded from the requirement to implement the PERR# signal and the Parity Error Response bit. • If a data phase parity error is detected, PERR# must be asserted (at the latest) in the second clock after completion of the data phase (i.e.. one clock after PAR is latched). Once PERR# is asserted, it must not be deasserted until during the third clock after the data phase completes. Figure 13-1 on page 204 and Figure 13-2 on page 208 both illustrate examples where the devices receiving the data checked parity and asserted PERR# before the completion of the data phase.

Parity error during Read

During a read transaction, the target sources the data and the parity. The initia- tor receives the data and parity and checks the parity for correctness. If the data is incorrect, the initiator must set the Detected Parity Error bit in its PC1 config- uration Status reglster (irrespective of the state of its Parity Error Response bit). Assuming that the initiator's Parity Error Response bit is set to one. the initiator asserts PERR# in the second dock following completion of the data phase and sets the Master Data Parity Error bit in its configuration Status register. Whether or not the bus master continues the transaction or terminates it is mas- ter design-dependent. The specifIcation recommends that the transaction be continued to completion. In addition to the assertion of PERR#, the bus master is required to report the parity error to the system software. The specification recommends utilization of an intenupt or the setting of a bit in a device-spedflc status register that is polled by the device driver. Alternately, the designer can assert SERRX, but this approach should not be used lightly It will more than likely result in a system shutdown.

Parity error during write

During a write transaction, the initiator sources the data and the parity. The tar- get receives the data and parity and checks the parlty for correctness. If the data is Incorrect, the target must set its Detected Parity Em bit to one (irrespective of the state of its Parity Ermr Response bit). Assuming that the target's Parity Ermr Response bit is set to one, the target asserts PERRX in the second clock fol- lowing completion of the data phase. The initiator samples PERRX on the sec- ond clock after completion of the data phase. If PERRX has been asserted by the target, the initlator sets the Master Data Parity Error bit in its PC1 configuration Status register. Targets never set this bit because only the bus master reports the error to software. Whether or not the bus master write continues the transaction or terminates it is mas- ter design-dependent. The specification remmmends that the transaction be continued to completion. In addition to the assertion of PERRt, the bus master is required to report the parity error to the system software. The specification recommends utilization of an interrupt or the setting of a bit in a devlce-specIflc status register that is polled by the device driver. Alternately, the designer can assert SERRX, but this approach should not be used lightly It will more than likely result in a system shutdown. In a particular platform design, the chip set logic may convert any assertion of PEW into SERRX.

Understand the usage of SERR# signal

SERR# is a required output from all PCI devices and is an input to the platform support logic (i.e.. the chipset). A PCI device is not permitted to assert SERR# unless the SERR# Enable bit in its configuration Command register is set to one. SERR# is implemented as an open-drain, shared signal because multiple PCI devices may assert SERR# simultaneously The system designer must provide a pull-up resistor on the SERR# signal line. • When asserted, SERR# is asserted starting on the rising-edge of the PCI clock and is asserted for one clock and then tri-stated. It may be asserted at any time (i.e. its assertion is not tied to any type or phase of a PCI transaction). The specification suggests that SERR# should be asserted as quickly as possible (within two clocks of detecting an error condition is recommended).

The SERR# signal is used to signal the follo wing types of conditions:

•parity error on the address phase of a tran saction

•parity error on the data phase of a Special Cycle transaction.

•error when attempting a memory write to deliver a message signaled interrupt

•serious problems other than parity detect ed by a PCI device.

•critical system failures detected by system board logic.

• In addition to the assertion of SERR#, the target device that is apparently addressed in a corrupted address phase can react in one of the following ways: • METHOD1 . assert DEVSEL# and complete the transaction normally. • METHOD 2. assert DEVSEL# and terminate the transaction with a Target Abort. • METHOD 3. not assert DEVSEL# and let the master time out and execute a Master Abort. • The target is not permitted to terminate the transaction with a Retry or a Disconnect.

Figure 13-5: Address Parity Generation/Checking

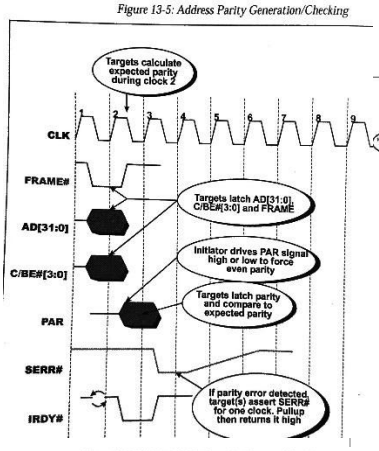


Figure 13-3: PCI Device's Configuration Command Register

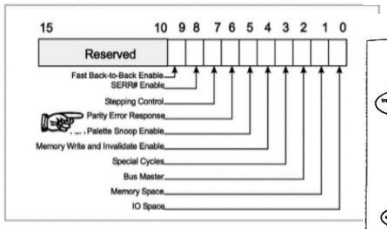
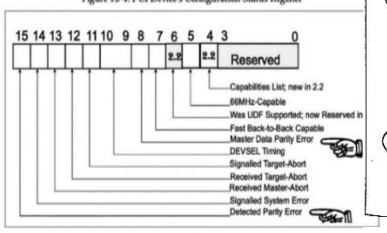


Figure 13-4: PCI Device's Configuration Status Register



64-bit Data Transfers and 64bit addressing

The PCI specification provides a mechanism that permits a 64-bit bus master to perform 64-bit data transfers with a 64-bit target. At the beginning of a transaction, the 64-bit bus master automatically senses If the responding target is a 64-bit or a 32-bit device. If it's a 64-bit device, up to eight bytes (a quadword) may be transferred during each data phase. Assuming a series of 0-wait state data phases, throughput of 264Mbytes/second can be achieved at a bus speed of 33MHz (8 bytes/transfer x 33 million transfers/second) and 528Mbytes/second at 66MHz. If the responding target is a 32-bit device, the bus master automatically senses this and steers all data to or from the target over the lower four data paths (AD[31:0]). The specification also defines 64-bit memory addressing capability. This capability is only used to address memory targets that reside above the 4GB address boundary. • It is important to note that 64-bit addressing and 64-bit data transfer capability are two features, separate and distinct from each other. A device may support one, the other, both, or neither.

Figure 15-3: Transfer Between a 64-bit Initiator and 64-bit Target

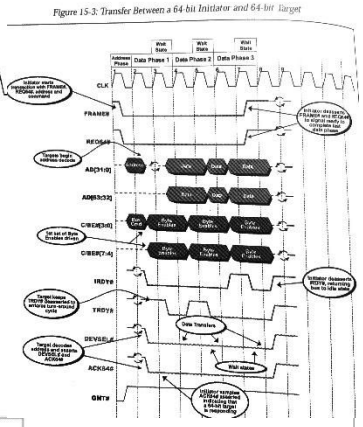


Figure 15-4: Transfer Between a 64-bit Initiator and a 32-bit Target

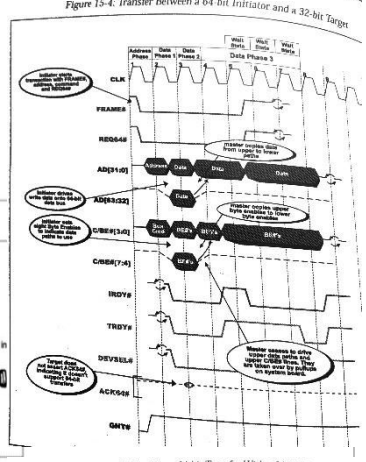


Figure 15-5: Single Data Phase 64-bit Transfer With a 64-bit Target

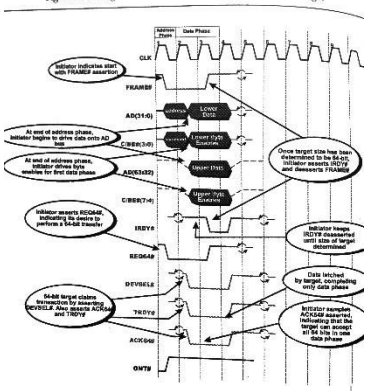


Figure 15-6: Dual Data Phase 64-bit Transfer With a 32-bit Target

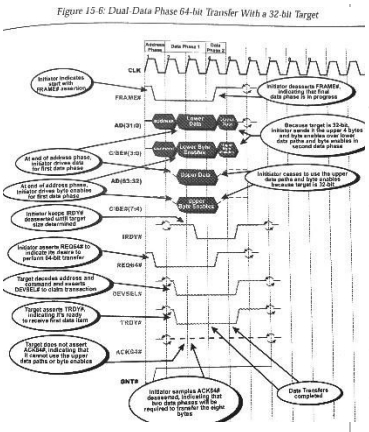


Figure 15-7: 32-bit Initiator Reading From Address at or Above 4GB

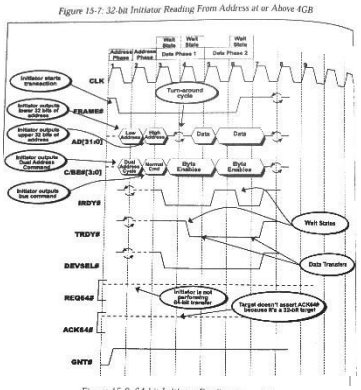
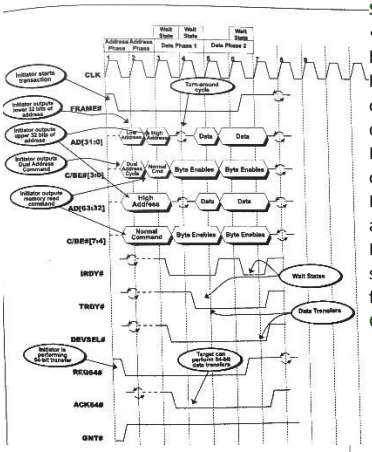


Figure 15-8: 64-bit Initiator Reading From Address Above 4GB With 64-Bit Data Transfers



33MHZ vs 66 MHZ PCI timing

• 66MHz components only operate correctly in a 3.3V signaling environment. The 5V environment is not supported. This means that 66MHz add-in cards are keyed to install only in 3.3V connectors and cannot be installed in 5V card connectors. The 66MHz PCI component or add-in card i ndicates its support in two fashions: progra mmatically and electrically.

How 66MHz components Determine Bus Speed

• When a 66MHz-capable device senses M66EN deasserted (at reset time). this automatically disables the device's ability to perform operations at speeds above 33MHz. If M66EN is sensed asserted, this indicates that no 33MHz devices are installed on the bus and the clock circuit is supplying a high-speed PCI clock.

Three Address Spaces: I/O, Memory and configuration

• PCI bus masters (including the host/PCI bridge) use PCI IO and memory transactions to access PCI IO and memory locations, respectively. • In addition, a third access type, the configuration access, is used to access a device's configuration registers. • A function's configuration registers must be initialized at startup time to configure the function to respond to memory and/or IO address ranges assigned to it by the configuration software.

PCI memory Space & PCI IO Space

The PCI memory space is either 4GB or 2^64 locations in size (if 64-bit addressing is utilized). • PCI IO space is 4GB in size (although Intel x86 processors cannot generate I0 addresses above the first 64KB of IO space).

PCI Configuration Space

• 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 referr ed to
as the function's configuration Header spac e. 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).

System with single PCI Bus

• The interface between the host processor bus and the PCI bus is referred to as the host/PCI bridge. • The bus directly on the other side of the bridge is always designated (for configuration purposes) as PCI Bus 0. • If one or more of the functions on PCI Bus 0 are PCI-to-PCI bridges, the two PCI buses connected to a PCI-to-PCI bridge are referred to as the bridge's primary (the PCI bus closer to the host processor) and secondary buses (the bus further away from the host processor).

Question 4

The goal of a cache is to reduce overall memory access time. Suppose that we are designing a cache and we have a choice between a direct-mapped cache where each row has a single 64-byte block of data, or a 2-way set associative cache where each row has two 32-byte blocks of data. Which one would you choose and why? Give a brief technical justification for your answer. If the choice would make no difference in performance then explain why not.

Solution:

Two blocks of 32-bytes each should give better performance. The main reason is that it will reduce conflicts between cache lines that have the same index bits but different tags. It also will probably avoid bringing in extra data in the wider cache line that is less likely to be used because it has lower spatial locality, and that will reduce the memory traffic.

Explain the advantages and disadvantages (in 4-5 sentences or a bulleted list) of using a direct-mapped cache instead of an 8-way set associative cache. Your answer must fit in the box below!

Answer

- A direct mapped cache should have a faster hit time; there is only one block that data for a physical address can be mapped to
- The above "pro" can also be a "con"; if there are successive reads to 2 separate addresses that map to the same cache block, then there may never be a cache hit. This will significantly degrade performance.
- In contrast, with a set associative cache, a block can map to one of 8 blocks within a set. Thus, if the situation described above were to occur, both references would be hits and there would be no conflict misses.
- However, a set associative cache will take a bit longer to search – could decrease clock rate.

6. Understand data/address parity generation, checking and parity error reporting on READ/WRITE transaction; Understand the usage of SERR# signal.
7. Understand 64-bit Data Transfers and 64-bit Addressing
8. Understand 33MHz vs. 66 MHz PCI Timing; Understand How Components Indicate 66MHz Support.
9. Understand PCI has three Address Spaces: I/O, Memory and Configuration