**Verilog Testbench**

module myf;

integer f, index, r;

reg [7:0] memory [0:15];

initial begin

$dumpfile("fdem.vcd");

$dumpvars(0, myf);

endA close up of a device

Description automatically generated

initial begin

f = $fopen("mem2.dat");

$display("Generating contents of file mem2.dat");

for(index = 0; index < 16; index = index + 1) begin

r = $random;

$fdisplay(f, "%b", r[12:5]);

end

$fclose(f);

$readmemb("mem2.dat", memory);

$display("\nContents of memory array: binary format");

for(index = 0; index < 16; index = index + 1)

$displayb(memory[index]);

repeat(2)

begin

$display("\nContents of memory array: hexdecimal format");

for(index = 0; index < 6; index = index + 1)

$displayh(memory[index]);

end

end

endmodule

**My cir**

`timescale 1 ns / 100 ps

module cir(a, b, f);

input a, b;

output f;

assign f = a | b;

endmodule

`timescale 1 ns / 100 ps

module cir\_tb;

reg a, b;

wire f;

cir g1 ( a, b, f);

initial begin

a = 0; b = 0;

#2 check\_res( f, 0 );

a = 0; b = 1;

#2 check\_res( f, 0 );

a = 1; b = 0;

#2 check\_res( f, 0 );

a = 1; b = 1;

#2 check\_res( f, 0 );

#10 $stop;

end

task check\_res;

input data;

input res;

begin

if( data != res) $display($time, "ns, Error: a=%b, b=%b, Expected value = %d, Actual value = %d \n", a, b, res, data);

end

endtask

**Design Principles for modern computer**

-All instructions directly executed by hardware

-Maximize rate at which instructions are issued

- Instructions should be easy to decode

-Only loads and stores should reference memory

-provide plenty of registers

**5 Stage Pipelining** A screenshot of a social media post

Description automatically generated

Stage 1: Instruction fetch Unit

Stage 2: Instruction decode Unit

Stage 3: Operand fetch unit

Stage 4: Instruction execution unit

Stage 5: Write back unit

**Superscalar architectures**

A screenshot of a social media post

Description automatically generated

Dual 5 stage pipelines w common instruction fetch unitA close up of text on a white background

Description automatically generated

Super scaler processor w 5 functional units

**PCI PROPERTIES**

|  |  |  |  |
| --- | --- | --- | --- |
| Bus  Type | Bus  Width | Bus  Speed | MB/sec |
| PCI | 32 bits | 33 MHz | 132 MBps |
| PCI | 64 bits | 33 MHz | 264 MBps |
| PCI | 64 bits | 66 MHz | 512 MBPS |
| PCI | 64-bits | 133 MHz | 1 GBps |

**PCI signals**

**AD:** AD[31:0] input/output. 32 bit address/data bus ,PCI is little endian (LSB)

Carries the start of the address. The resolution of this address is on doubleword boundary during a memory or configuration transaction.

The data bus is driven by the intiator during a write or the currently addressed target (during a write)

**C/BE:** Command or Byte Enable bus C/BE[3:0] defines the type of transaction 4bit command/byte enable bus. Indicates bytes enable during data phase

**-**Each bit corresponds to a byte-lane in AD[31:0] for ex, C/BE#[0] is the byte enable for AD[7:0]

**Parity Signal, PAR:**

-parity bit, used to verify correct transmittal of address/data and command/byte-enable

-The XOR of AD[31:0], C/BE#[3:0], and PAR should return zero (even partity)

The number of 1’s across these 37 signals should be even.

-is driven by either the intitator (during a write) or the currently addressed target(during a read) one clock after the assertion of #IRDY

**FRAME#:** signals the start and end of a transaction

Low active

**TRYDY#:** Target ready, when the target asserts this signal, it tells the initiator that it is ready to send or receive data

**IRDY#:** Initiator ready, Assertion by initiator indicates that it is ready to send receive data

**STOP#:** used by target to indicates that it needs to terminate the transaction.

**DEVSEL#:** Device Select, part of PCI distributed address decoding.

Each target is responsible for decoding the address associated with each transaction

When a target recognizes its address it asserts DEVSEL# to claim the corresponding transactionA screenshot of a cell phone

Description automatically generated

Is asserted by a target when the target has decoded its address

**CLK:** PCI input clock, all signals sampled on rising edge , 33MHz is really 33.33333MHz (30ns clk. Period). The clock is allowed to vary from 0 to 33 MHz

**RST#:** Asynchronous reset -PCI devices must tri-state all I/Os during reset

**PCI Bus Arbitration**

A screenshot of a cell phone

Description automatically generatedA close up of a map

Description automatically generated

**Bus Arbitration** refers to the process by which the current bus master accesses and then leaves the control of the bus and passes it to the another bus requesting processor unit

Centralized arbitration – a single bus arbiter performs the required arbitration

PCI Bus Arbiter -uses centralized arbitration

-independent grant and request lines

- REQ# and GNT # lines for each device

-one or more PCI master devices may require use of the PCI bus to preform a data transfer with another PCI device

-does not specify a particular policy but Mandates the use of fair policy

**Fairness:** The central arbiter is required to implement a fairness algorithm to avoid deadlocks. Fairness means that each potential bus master must be granted access to the bus independent of other requests. Fairness is defined as a policy that ensures that high priority masters will not dominate the bus to the exclusion of lower priority masters when they are continually requesting the bus. However, this does not mean that all the agents are required to have equal access to the bus.

EX) The order in which the master would receive access to the bus is :

1.Master A , 2. Master B, 3. Master X, 4. Master A , 5. Master B, 6 Master Y, 7. Master A , 8. Master B , 9. Master Z , 10. Master A , 11. Master B, 12. Master X, etc

A close up of a logo

Description automatically generated

**4 PCI STATES:**

**PCI BUS arbitration between 2 masters**

**PCI Bus Parking:**

-A master must only assert its REQ# to signal a current need for the bus

-if a system designer implements a bus parking scheme, a default bus owner should be defined when no other masters request bus

-the choice of which master to park the bus on is defined by the designer of the bus arbiter.

**Hidden Bus Arbitration**

The PCI scheme allows bus arbitration to take place while the current initiator is preforming a data transfer. If the arbiter decides to grant ownership of the bus for the next transaction to a master other than the initiator of the current transaction, it removes the GNT# from the current initiator and issues GNT# to the next owner of the bus. The next owner cannot assume bus ownership, however, until the bus is idled by the current initiator. No bus time is waisted on a dedicated period of time to preform an arbitration cycle.

**PCI arbitrator request/grant timing** A close up of a newspaper

Description automatically generated

**Latency** A close up of a newspaper

Description automatically generated

**Timing requirement to prevent master from monopolizing the bus.**

Master must trasnfer data within 8 clock cycles. It is a rule that the initiator must keep IRDY# deasserted for more than seven PCI clocks during any data phase. In other words, it must be prepared to transfer a data item within eight clocks after entry into any data phase.

**Master Latency Timer(LT)**

Minimum amount of time (in PCI clock periods) that the bus master is permitted to retain ownership of the bus each time that it acquires bus ownership and initiates a transaction .

**Bus Accesss latency**

Defined as the amount of time that expires from the moment a bus master requests the use of the PCI bus until it completes the first data transfer of the transaction. In other words, it is the sum of the arbitration, bus acquisition and target latency

**Arbitration Latency**

Defined as the period of time from the bus masters assertion of REQ# until the bus arbiter asserts the bus master’s GNT#. This period is a function of the arbitration algorithm, the masters priority and wether any other masters are requestiing access to the bus.

**Bus Acquisition Latency**

Defined as the period time from the reception of GNT# by the requesting bus master until the current bus master surrenders the bus. The requesting bus master can then inititate its transaction by asserting FRAME#. The duration of this period is a function of how long the current bus master’s transaction in progress takes to complete. This parameter is the larger of either the current masters LT value(in other words its timeslice) or longest latency to first data phase completion in the system(which is limited ti a maximum of 16 clocks).

**Initiator and target ready**

Defined as the period of time from the start of a transaction until the master and the currently addressed target are ready to complete the first data transfer of the transaction.This period is a function how fast the master is able to transfer the first data item,as well as the access time for the currently addressed target device(and is limited to a maximum of 8 clocks for the master and 16 clocks for the target).

**Bus Latency Defined**

Is defined as as the amount of time that expires from the moment a bus master requests the use of the PCI bus until it completes the first data tansfer of the transaction.

**LT Hardwired**

Yes, for master that preforms one or two data phases per transaction, but the hardwired value may not exceed 16(and it could be zero).

**General**

The problem of a bus master hogging the bus is solved by:

1.The inclusion of the LT associated with each master.

2. The rule that requires the initiator to keep IRDY# deasserted for no longer than eight PCI clocks during any data phase.

Preventing Target from Monopolizing Bus

Case 1. The target cannot transfer the first data item within 16 clocks from the assertion of FRAME#.

Case 2. Although the target can transfer the first data item within 16 clocks from the assertion of FRAME#, it cannot transfer one of the subsequent data items within 8 clocks from the start of the respective data phase. Any data phase other than the first one is referred to as a subsequent data phase.

**Demo.v and Demo\_tb**

module demo (count, count\_tri, clk, rst\_l, load\_l, enable\_l, cnt\_in, oe\_l);

output [3:0] count;

output [3:0] count\_tri;

input clk;

input rst\_l;

input load\_l;

input enable\_l;

input [3:0] cnt\_in;

input oe\_l;

reg [3:0] count;

// tri-state buffers

assign count\_tri = (!oe\_l) ? count : 4'bZZZZ;

// synchronous 4 bit counter

always @ (posedge clk or negedge rst\_l)

begin

if (!rst\_l) begin

count <= #1 4'b0000;

end

else if (!load\_l) begin

count <= #1 cnt\_in;

end

else if (!enable\_l) begin

count <= #1 count + 1;

end

end

endmodule

`timescale 1 ns / 100 ps

module demo\_tb;

reg clk\_50;

reg rst\_l, load\_l, enable\_l;

reg [3:0] count\_in;

reg oe\_l;

wire [3:0] cnt\_out;

wire [3:0] count\_tri;

demo U1 ( .count(cnt\_out), .count\_tri(count\_tri), .clk(clk\_50),

.rst\_l(rst\_l), .load\_l(load\_l), .cnt\_in(count\_in),

.enable\_l(enable\_l), .oe\_l(oe\_l)

);

// create a 50Mhz clock

always

#10 clk\_50 = ~clk\_50; // every ten nanoseconds invert

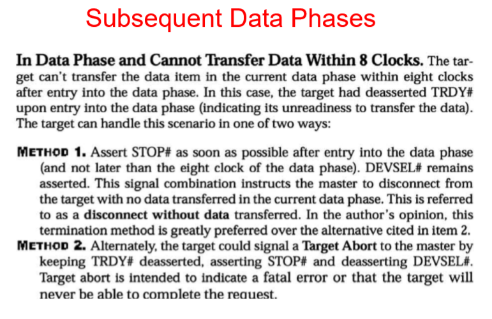
initial begin

$display($time, " << Starting the Simulation >>");

clk\_50 = 1'b0;

rst\_l = 0; enable\_l = 1'b1; load\_l = 1'b1; count\_in = 4'h0; oe\_l = 4'b0;

#20 rst\_l = 1'b1;

$display($time, " << Coming out of reset >>");

@(negedge clk\_50); // wait till the negedge of

load\_count(4'hA);

@(negedge clk\_50);

$display($time, " << Turning ON the count enable >>");

enable\_l = 1'b0;

wait (cnt\_out == 4'b0001);

$display($time, " << count = %d - Turning OFF the count enable >>", cnt\_out);

enable\_l = 1'b1;

#40;

// let the simulation run for 40ns

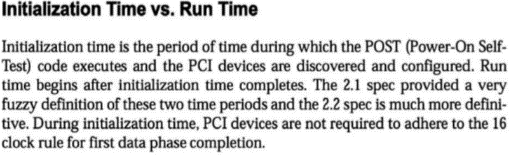
// the counter should not count

$display($time, " << Turning OFF the OE >>");

oe\_l = 1'b1;

// disable OE, the outputs of count\_tri should go high Z.

#20;

$display($time, " << Simulation Complete >>");

$stop;

end

initial begin

// $monitor will print whenever a signal changes in the design

$monitor($time, " clk\_50=%b, rst\_l=%b, enable\_l=%b, load\_l=%b, count\_in=%h, cnt\_out=%h, oe\_l=%b, count\_tri=%h", clk\_50, rst\_l,enable\_l, load\_l, count\_in, cnt\_out, oe\_l, count\_tri);

end

// The load\_count task loads the counter with the value passed

task load\_count;

input [3:0] load\_value;

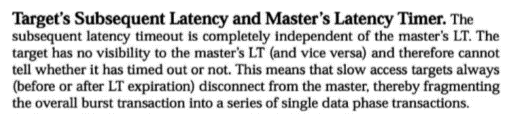
begin

@(negedge clk\_50);

$display($time, " << Loading the counter with %h >>", load\_value);

load\_l = 1'b0;

count\_in = load\_value;

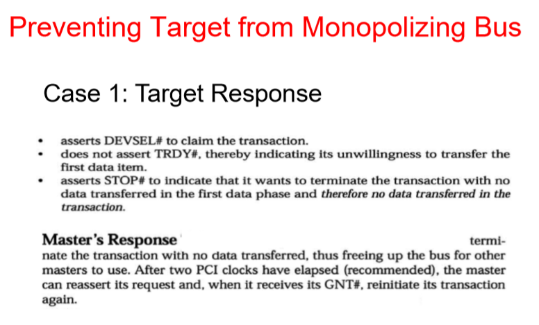
@(negedge clk\_50);

load\_l = 1'b1;

end

endtask

endmodule

**Latency Continued** 

**Two Exceptions of first Data Phase Rule**

-During system initialization time (defined as 2^25 CLKs after RST# removed), targets do not have to adhere to the 16 clock rule. After initialization time has elapsed, all targets must obey the 16 clock rule for first data phase completion

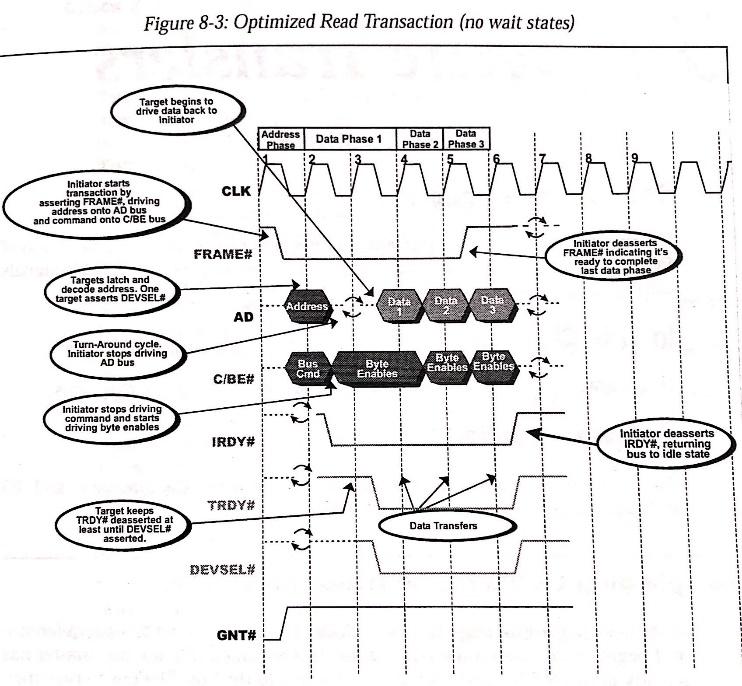
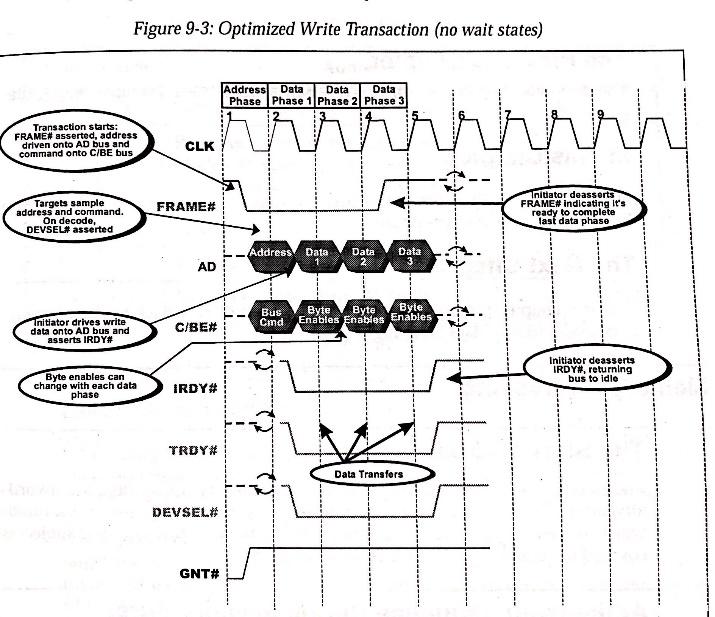
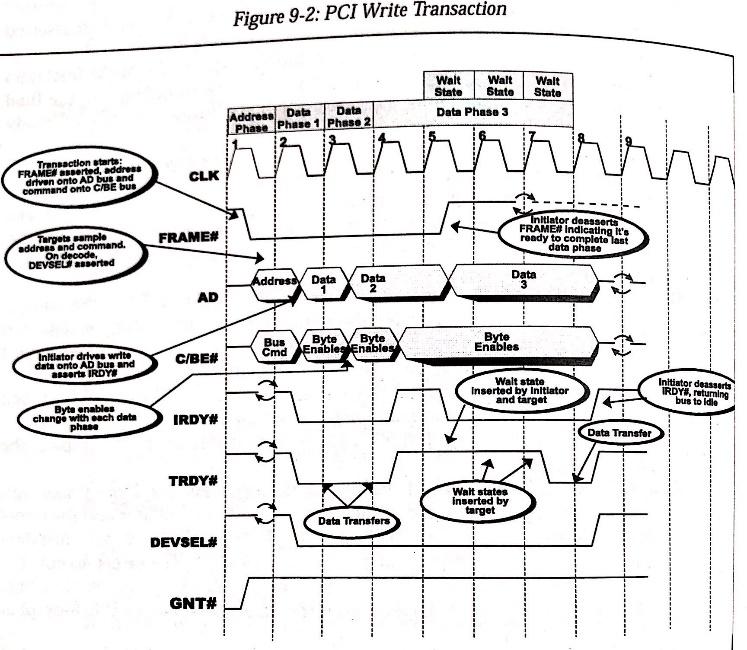
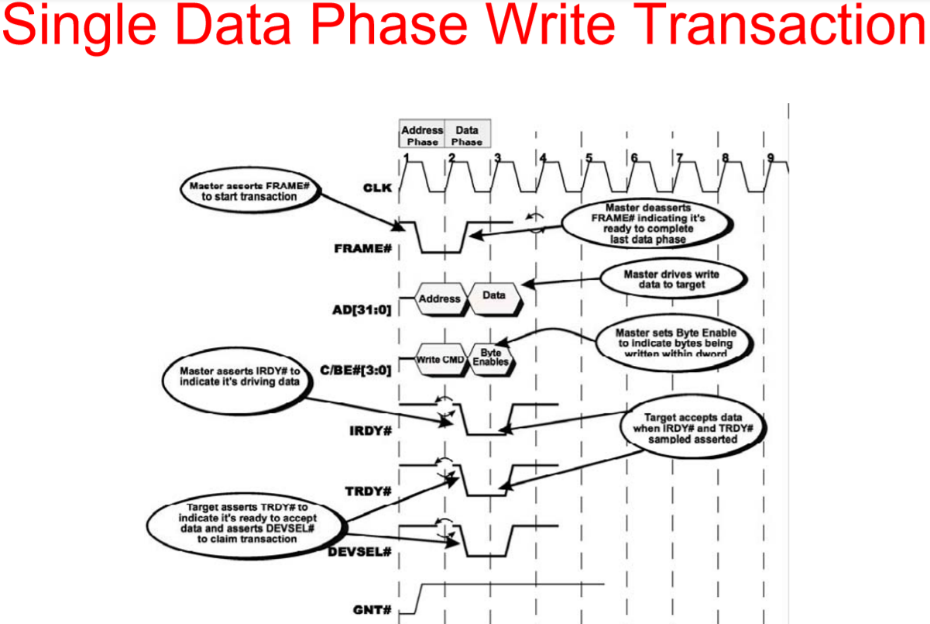
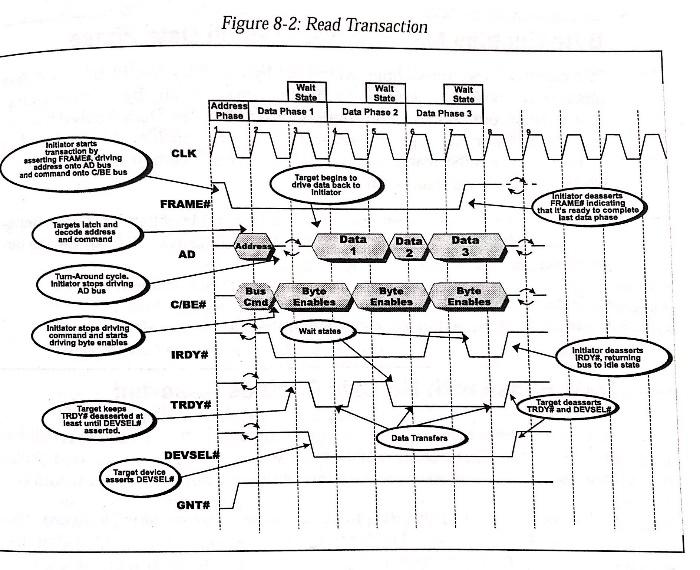
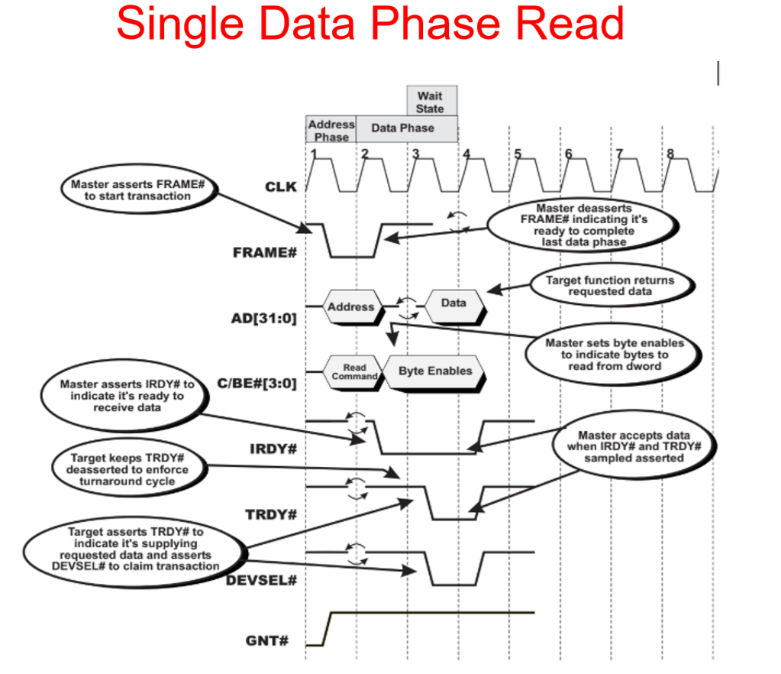
-A host/PCI bridge that is snooping is permitted to exceed the 16 clock limit but may never exceed 32 clocks.

**Subsequent Data Phases**

**General** If it will take more than eight clocks to complete a data phase other than the first (referred to as the subsequent latency timeout) and it is not the final data phase (FRAME# is still asserted), the target must force the master to terminate the transaction

**Master’s Response**

The master must terminate the transaction. After two PCI clock have elapsed, the master that received the disconnect can reassert it request and, when it receives its GNT#, Reinitiate its transaction again at the next data item.



**Performance during read transactions**

**Be able to discuss performance during**

A turn-around cycle must be included in the first data transfer of a read transaction.

A single data phase read from a target always consists of at least three clock cycles (one dock cycle for the address phase and two clock cycles for the data phase).

At a clock rate of 33MHz, a read transaction consisting of a single data transfer would take 90ns to complete. An idle cycle (30ns in duration at 33 MHz) must be included between transactions, resulting in 120 ns per transaction.

**Back-to-back single data phase read transfers**

Using back to back single data phase read transfers, the data throughput would be 8.33 million transfers per second. If each transfer involved four bytes, the resultant transfer rate would be 33.33 Mbytes per second.

-In actual practice, most read transactions Involve a burst transfer of multiple objects (words or quadwords) between the initiator and the currently addressed target

-The read transaction involving multiple data phases only requires the turn around cycle during the first data phase.

**The achievable burst transfer rate during the second through the last data read phases**

The second through last data phases can each be accomplished in a single clock cycle (if both the initiator and the currently-addressed target are capable of zero wait state transfer).

The achievable transfer rate during the second through the last data phases is thus one transfer every 30ns (at PCI bus speed of 33MHs) or 33 million transfers per second. If each data phase involves the transfer of four bytes, the resultant data transfer rate is 132 Mbytes per second.

**Performance During Write Transactions**

Transactions where in an initiator a single data phase write to a target consist of at least two cycles of the PCI clock (The address phase and a one clock data phase).

An idle cycle (at 33MHz, 30 ns in duration) must be included between transactions (except when preforming a fast back to back transaction pair)

**Back-to-back single data phase write transfers**

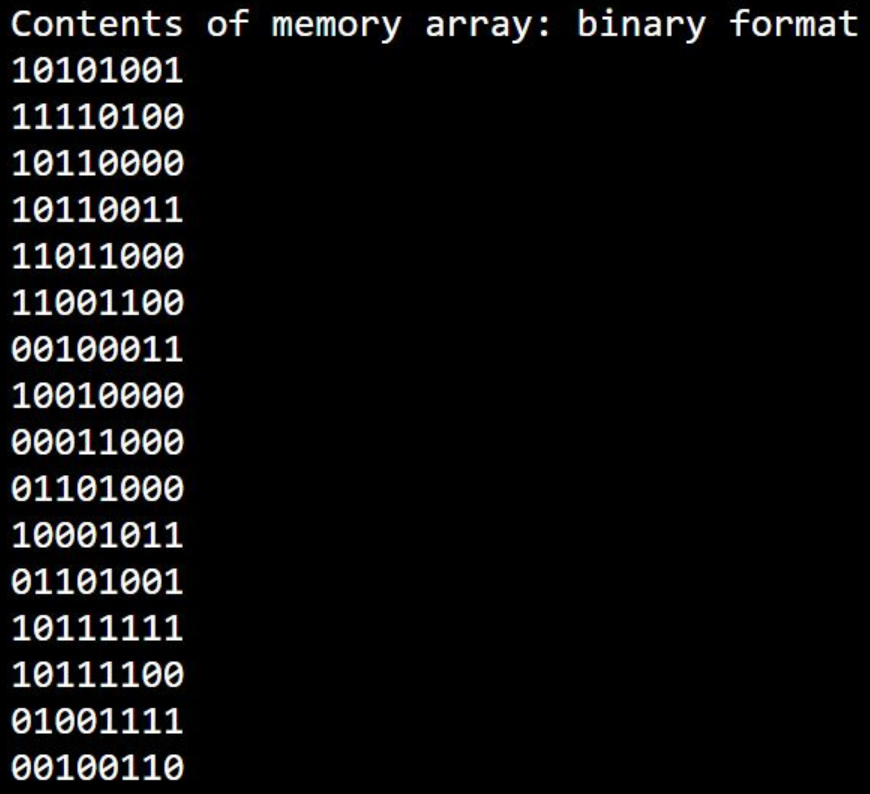
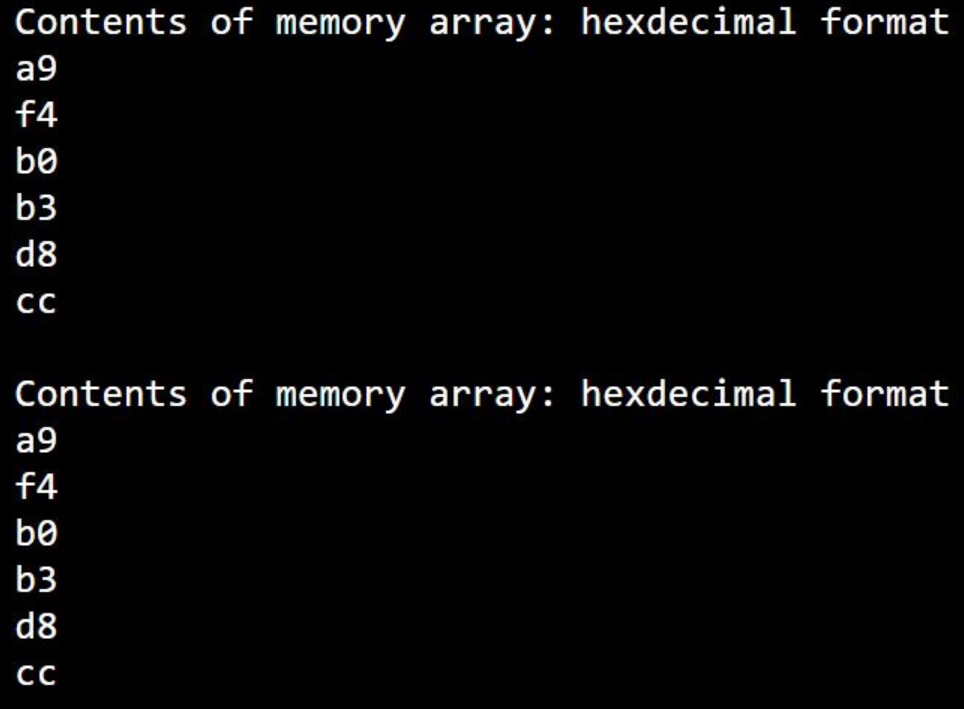
At a clock rate of 33 MHz, a single data phase write transaction takes 90ns to complete. Using back-to-back single data phase write transfers, the data throughput would be 11.11 million transfers per second. If each transfer involved four bytes, the resulting transfer rate would be 44.44 Mbytes per second.

**The achievable burst transfer rate during the second through the last data write phases**

The first through the last data transfer of a burst write transaction can each be accomplished in a single clock cycle (if both the initiator and the currently addressed target are capable of zero wait state data phases).

The achievable transaction rate during the first through the last data phases is thus one transfer every 30ns(at a PCI bus speed of 33 MHz) or 33 million transfer per second. If each transfer involves the transfer of four bytes, the data transfer rate is 132 Mbytes per second.

**MISC**

Syntax

task task\_name;

[declaration]

procedural\_statement

endtask

Task is defined inside a module

\* Tasks may have delays

\* Task may have zero or more input, output and inout arguments

task task\_name;

parameter\_declaration;

input\_declaration;

output\_declaration;

inout\_declaration;

register\_declaration;

event\_declaration;

statement;

endtask

