PCI Project Report

Submitted by:

Name	Section
Zyad Mohamed Abd-El-Aziz Ali	2
Samir Mosaad Ibrahim Mosaad	2
Abdelrahman Shawqy Ali El-sayed	2
Reham Ashraf El-sayed Ali shouman	2
Rita Ihab Mouris Abd-El-kodous Al-Nomier	2
Nada Ihab Ahmed Mohamed Abd-El-Gawad	3

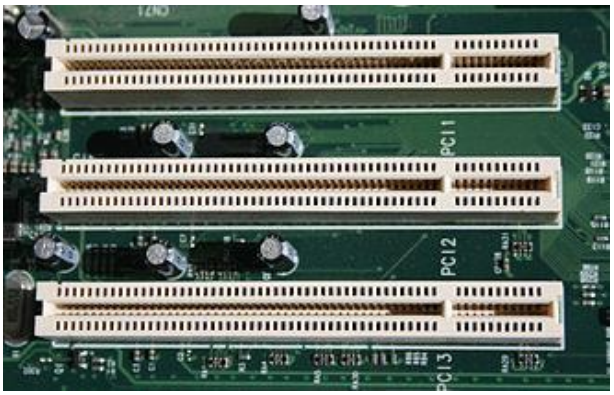
Cairo, Egypt

2018/2019

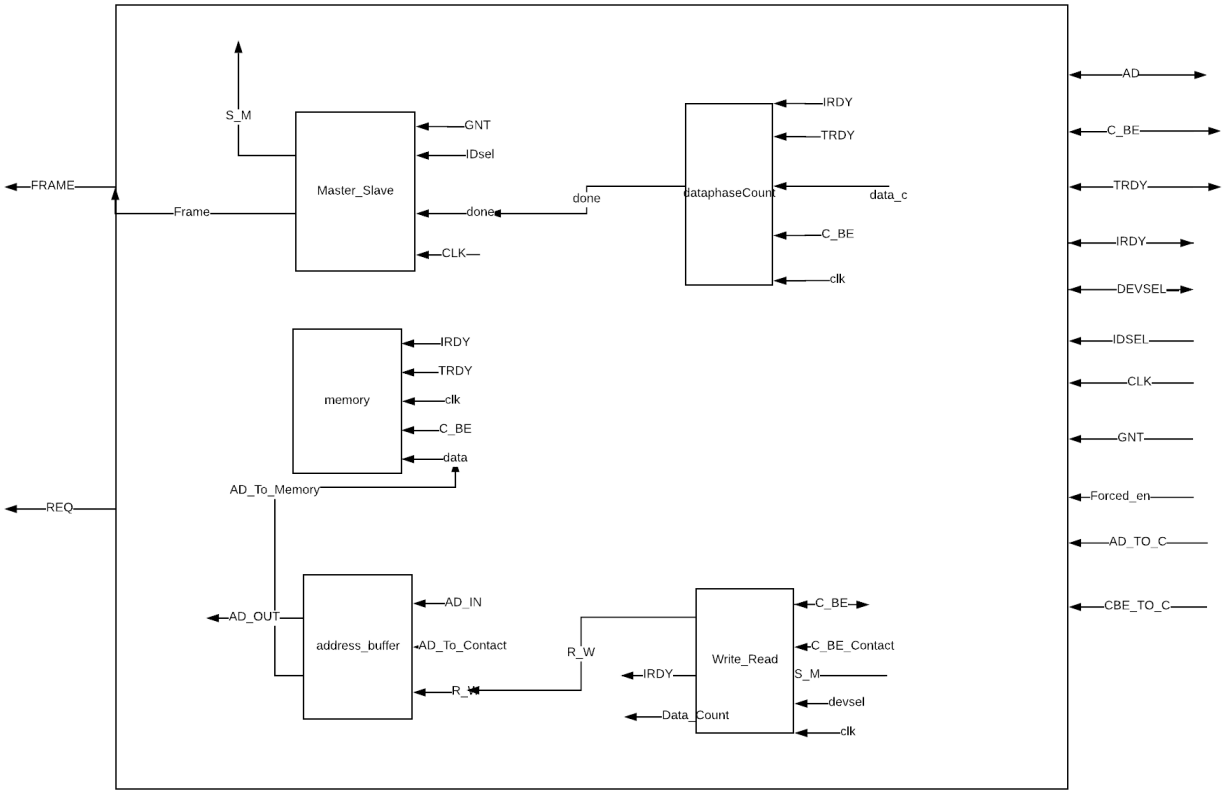
1. Table of contents

2. Introduction

Conventional PCI, often shortened to **PCI**, is a local computer bus for attaching hardware devices in a computer. PCI is the acronym for **Peripheral Component Interconnect** and is part of the PCI Local Bus standard. The PCI bus supports the functions found on a processor bus but in a standardized format that is independent of any particular processor's native bus. Devices connected to the PCI bus appear to a bus master to be connected directly to its own bus and are assigned addresses in the processor's address space. It is a parallel bus, synchronous to a single bus clock.

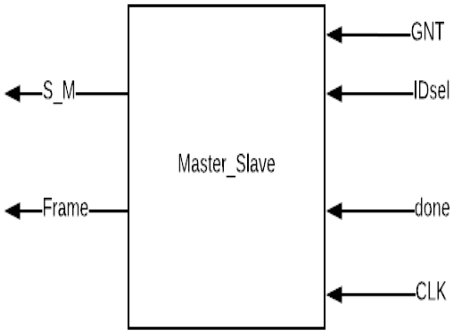


3. Device Module



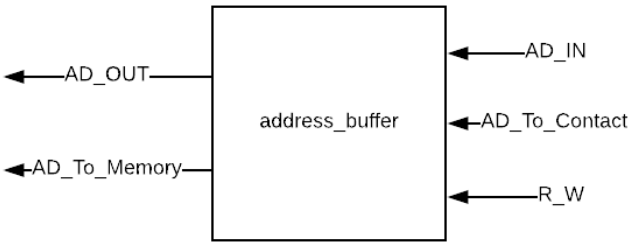
During the designing phase of our code of the PCI protocol, the device (initiator/target) was the most challenging part so we thought about dividing it to small but easy parts and we followed a hierarchical structure.

3.1. Master Slave Module



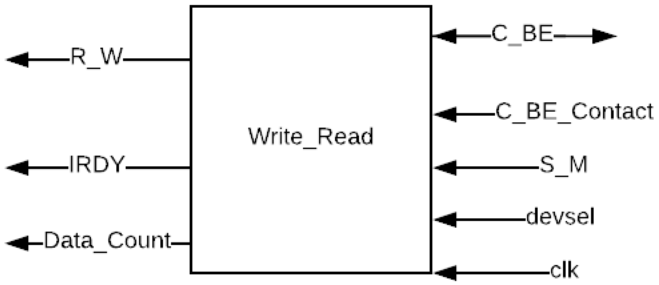
Inputs	GNT (Grant) IDSEL (Target Select) DONE (Flag for the frame) CLK (System clock)
Outputs	S_M (1 for master and 0 for slave) Frame
Description	Synchronous module that checks whether the device is master or slave, if the device has taken the grant and the target select signal (IDSEL) is high then the device is an initiator, if the device has not taken the grant and the target select signal (IDSEL) is low then the device is a target.

3.2. Address Buffer Module



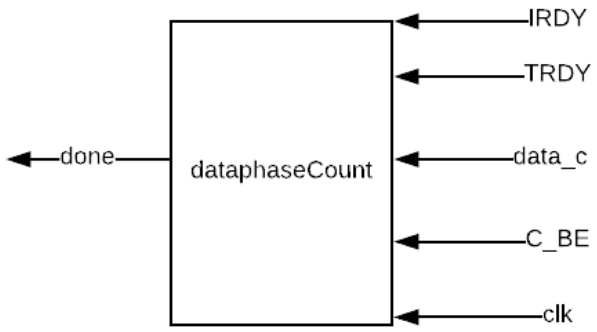
Inputs	AD_IN (Connected to AD lines) AD_To_Contact (Forced input) R_W (1 for write and 0 for read)
Outputs	AD_OUT AD_To_Memory (Data to be written in memory)
Description	Asynchronous module that acts as a tristate buffer, if R_W signal is high means that the initiator wishes for a write transaction so AD_IN is connected to AD_To_memory which goes directly to memory, if R_W signal is low means that the initiator wishes for a Read transaction.

3.3. Write Read Module



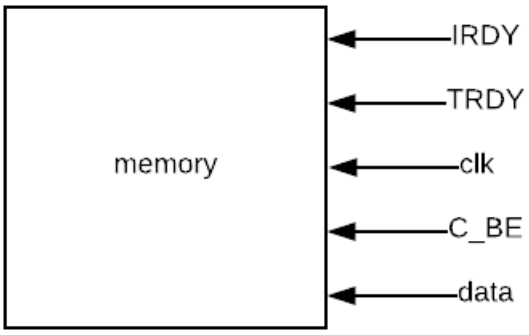
Inputs	C_BE C_BE_Contact (Forced input) S_M (1 for master and 0 for slave) Devsel (if it's 0 then the needed target is selected) Clk (system clock)
Outputs	R_W (1 for write and 0 for read) IRDY Data_Count (if it's 1 means that we are in the data phase)
Description	Synchronous module that checks whether we are in the address phase or the data phase according to the first two bits of C_BE [1:0], if they are zeroes then we are in the data, the last two bits of C_BE [3:2] are the number of data phases, if C_BE [1:0] are not zeroes that means that we are in the address phase so it checks whether the command written in C_BE is a read or a write command.

3.4. Data Phases' count Modules



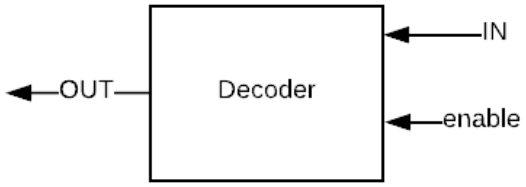
Inputs	IRDY (Initiator ready) TRDY (Target ready) Data_C (if it's 1 means that we are in the data phase else we are in the address phase) C_BE [3 : 2] (the last two bits are the number of data phases) CLK (system clock)
Outputs	Done (flag for the frame)
Description	Synchronous module in which it checks whether the device is in the data phase or not, if yes then it initialize the counter with the number of data phases (last two bits in C_BE [3:2]), then the counter is decremented by 1 when TRDY and IRDY both are low and before reaching the last data phase, the frame is deasserted.

3.5. Memory module



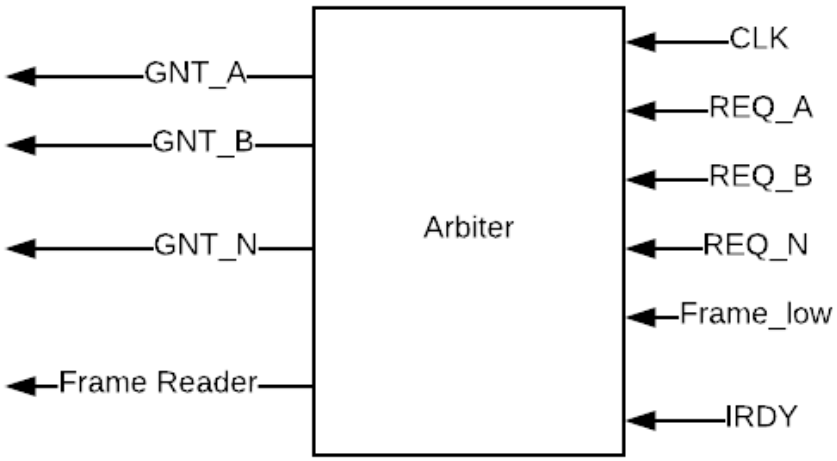
Inputs	IRDY (Initiator ready) TRDY (Target ready) CLK (System clock) C_BE () Data (during write transaction, the data coming from the is reserved here)
Outputs	NONE
Description	Synchronous module in which it takes data in the slave mode through the AD bus from the address buffer module.

4. Decoder Module



Inputs	IN (the last two bits of the address bus) Enable (turn the decoder on or off)
Outputs	OUT (outputs the IDsel)
Description	Asynchronous module that is operated if the enable signal is high, it takes the address of the needed target to contact as an input then it outputs the IDSEL signal of this target.

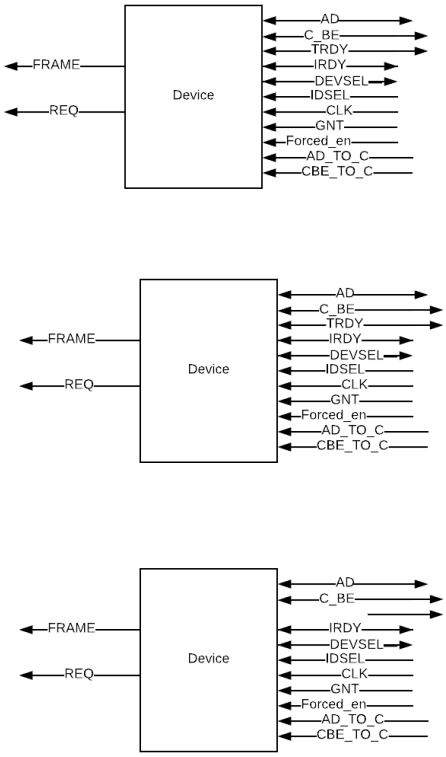
5. Arbiter

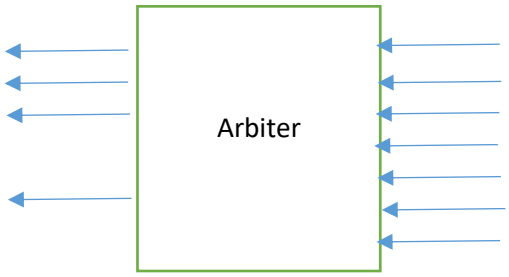


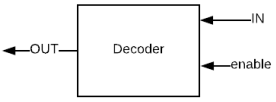
Inputs	CLK (System clock) REQ_A(Request from device A) REQ_B(Request from device B) REQ_N(Request from device N) Frame_low(Frame) IRDY(Initiator ready)
Outputs	GNT_A(Grant given to device A) GNT_B(Grant given to device B) GNT_N(Grant given to device N) Frame_Reader(Signal used for the Frame)
Description	Synchronous module that operates according to priority mode, the arbiter stores all the coming requests in an internal memory then it gives grants according to highest priority device, it checks whether the bus is idle or not, in case it's idle then it gives the grant to the highest priority device, then it checks whether the bus is taken or not and which device has taken it to remove it's grant.

6. Test bench results

6.1. Top module





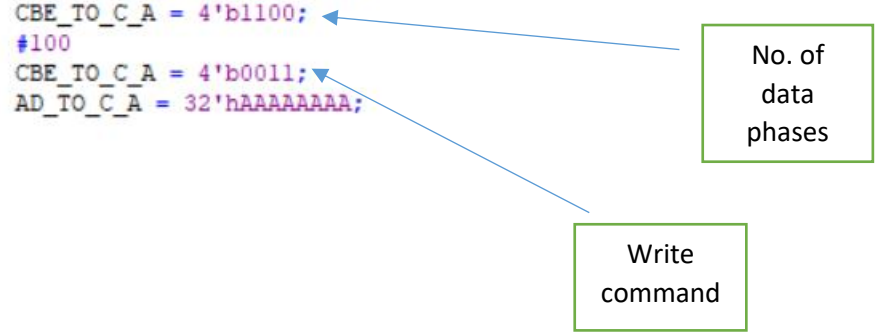


6.1. Scenarios

Here in the test bench we tried out the three different scenarios mentioned in the description.

In the first scenario, device A requests the bus in order to communicate with device B, and it wishes for three data phases in one transaction.

```
clk = 0;
#200

forced_en_A = 0;
#100
forced_en_A = 1 ;
#200
AD_TO_C_A = 32'h00000001;
CBE_TO_C_A = 4'b1100;
#100
CBE_TO_C_A = 4'b0011;
AD_TO_C_A = 32'hAAAAAAAA;
```



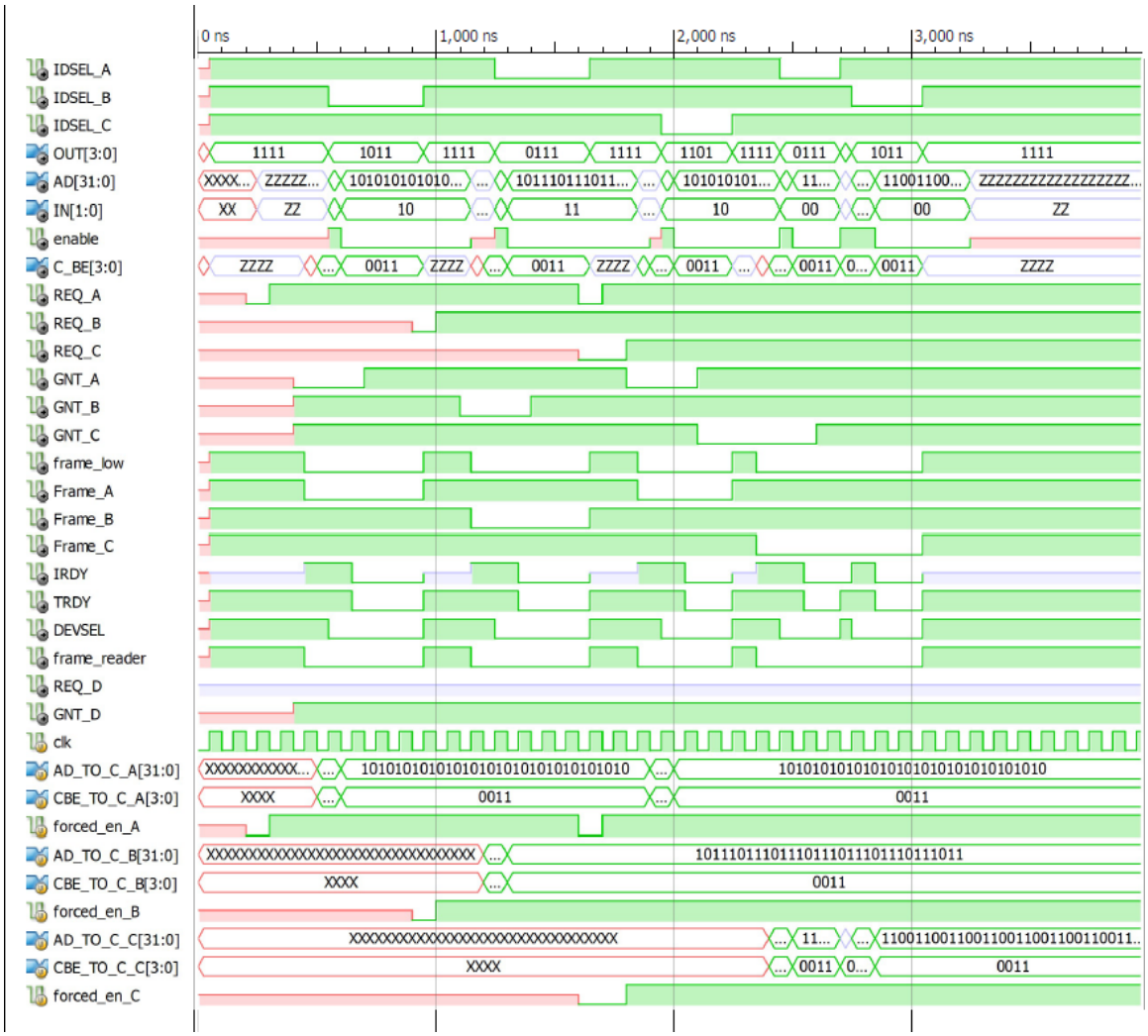
In the second scenario, device B requests the bus in order to communicate with device A, and it wishes for two data phases in one transaction.

```
#300

forced_en_B = 0;
#100
forced_en_B = 1 ;
#200
AD_TO_C_B = 32'h00000000;
CBE_TO_C_B = 4'b1100;
#100
CBE_TO_C_B = 4'b0011;
AD_TO_C_B = 32'hBBBBBBBB;
```

In the third scenario, both device A and device C request the bus at the same time, but the arbiter will give the grant to device A due to its high priority, device A will send two data words to device C, then the arbiter will give the grant to device C, then device C will send one data word to device A, then it will communicate with device B and send another data word.

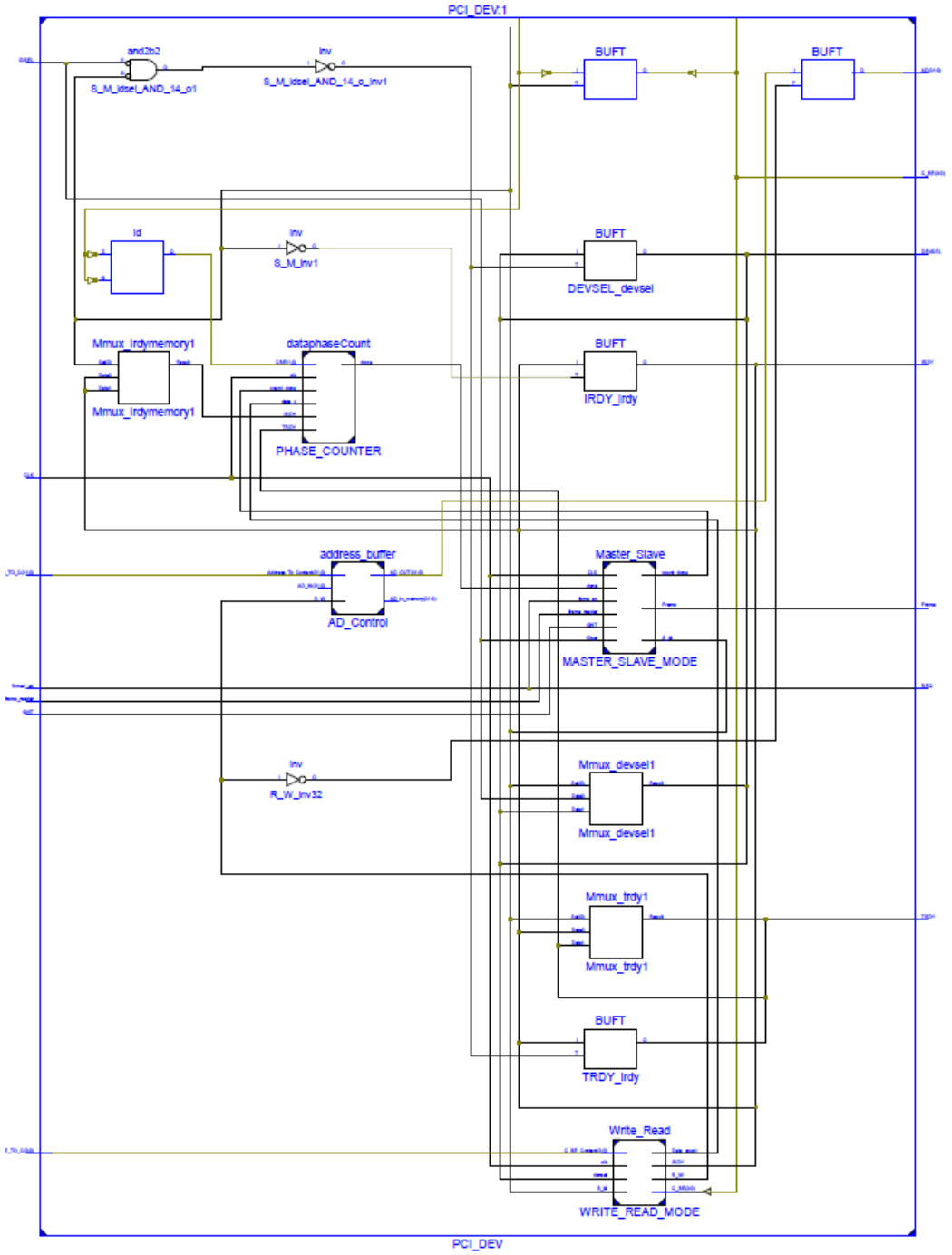
```
#300

forced_en_A = 1'b0;
forced_en_C = 1'b0;
#100
forced_en_A = 1'b1;
#100
forced_en_C = 1'b1;
#100
AD_TO_C_A = 32'h00000002;
CBE_TO_C_A = 4'b1000;
#100
AD_TO_C_A = 32'hAAAAAAAA;
CBE_TO_C_A = 4'b0011;
#400
AD_TO_C_C = 32'h00000000;
CBE_TO_C_C = 4'b0100;
#100
AD_TO_C_C = 32'hCCCCCCCC;
CBE_TO_C_C = 4'b0011;
#200
AD_TO_C_C = 32'b0000_0000_0000_0000_0000_0000_0000_00zz;
CBE_TO_C_C = 4'b0100;
#50
AD_TO_C_C = 32'h00000001;
#100
AD_TO_C_C = 32'hCCCCCCCC;
CBE_TO_C_C = 4'b0011;
```

6.2. Wave


0 ns
1,000 ns
2,000 ns
3,000 ns
IDSEL_A
IDSEL_B
IDSEL_C
OUT[3:0]
AD[31:0]
IN[1:0]
enable
C_BE[3:0]
REQ_A
REQ_B
REQ_C
GNT_A
GNT_B
GNT_C
frame_low
Frame_A
Frame_B
Frame_C
IRDY
TRDY
DEVSEL
frame_reader
REQ_D
GNT_D
clk
AD_TO_C_A[31:0]
CBE_TO_C_A[3:0]
forced_en_A
AD_TO_C_B[31:0]
CBE_TO_C_B[3:0]
forced_en_B
AD_TO_C_C[31:0]
CBE_TO_C_C[3:0]
forced_en_C

7. Circuits

7.1. Device circuit



7.2. Arbiter circuit