Lab Report: Project 1
PCI Arbiter

Team 3:

Sam Lee
Miguel Tirado
Jesus Nunez
Muhammad Mujeeb
Angel Smith-Evans

CPE 186: Computer Hardware System Design
Fall 2019

Professor Jing Pang

Table of Contents

1. Introduction

The PCI bus arbiter performs bus arbitration among multiple masters on the PCI bus. Suppose 3 bus masters reside on the PCI bus, as well as requests for the bus. One pair of request and grant signals is dedicated to each bus master. A bus arbiter is a mechanism that controls the flow of data within a system. An arbiter controls the flow of data by accepting request signals from PCI devices and will grant permission to transfer data within the PCI bus. By itself an arbiter would not be able to effectively perform data transfers as the default algorithm for request signals is on a first come first serve basis. In order to avoid ineffective priority assessments, we implemented an algorithm called round robin. In the rotating priority scheme, the requestor that is most recently granted the bus receives the lowest priority, while the requestor position next to it receives the highest priority. The remaining requestor receives subsequently lower priority based on its position. This algorithm will start a countdown timer for each approved grant signal. Moreover, while data transferring is occurring, another request signal will be approved, however, its respective grant signal will not be approved until the countdown timer has ended or the current grant signal has finished its data transfer. By implementing the round robin algorithm into the arbiter design, we hope to see efficient data transfers.

2. PCI Arbiter

A. Design Purpose

The purpose of this design is to implement a fully functioning PCI Arbiter using Verilog HDL with methodology learned in our class, CPE 186. Our purpose for learning how to implement this design was to understand at an intimate level how the PCI Arbiter grants permissions to allow for multiple masters to be able to write to the data bus using round robin scheme. Each master device would have one pair of REQ# and GNT# signals and the bus arbiter would perform arbitration among these devices using the round-robin arbitration scheme. In the Round-Robin scheme, the requestor that is most recently granted the bus receives the lowest priority, while the requestor position next to it receives the highest priority. We then started our design by drawing our Finite State Machine.

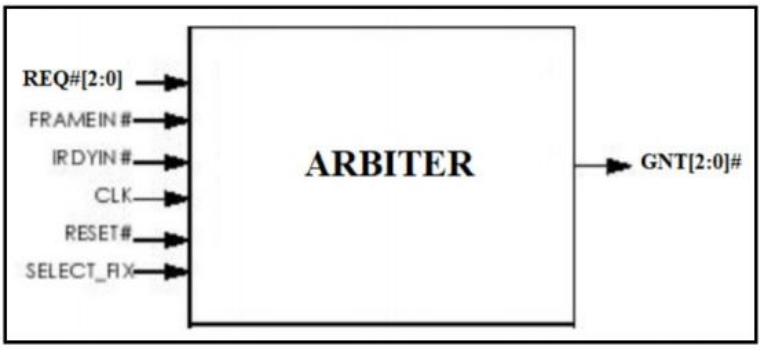

Fig 1. PCI Arbiter Circuit Diagram

B. Engineering Data:

This specific arbiter design shows that we have 6 input signals and one output signal. We decided to design arbiter using a Finite State Machine (FSM) as per the instructions were given to us because it allows us to work with between modules or subsystems. Figure 1 shows the schematic drawing of a PCI Arbiter.

C. PCI Arbiter State Diagram

Below is the Finite State Diagram that was used to implement our Verilog code for the PCI Arbiter design. There are four states: a, b, c, and idle. The state machine will cycle through the states based on the req[2:0] signal. Like the rest of the PCI arbiter, it is a low active signal.

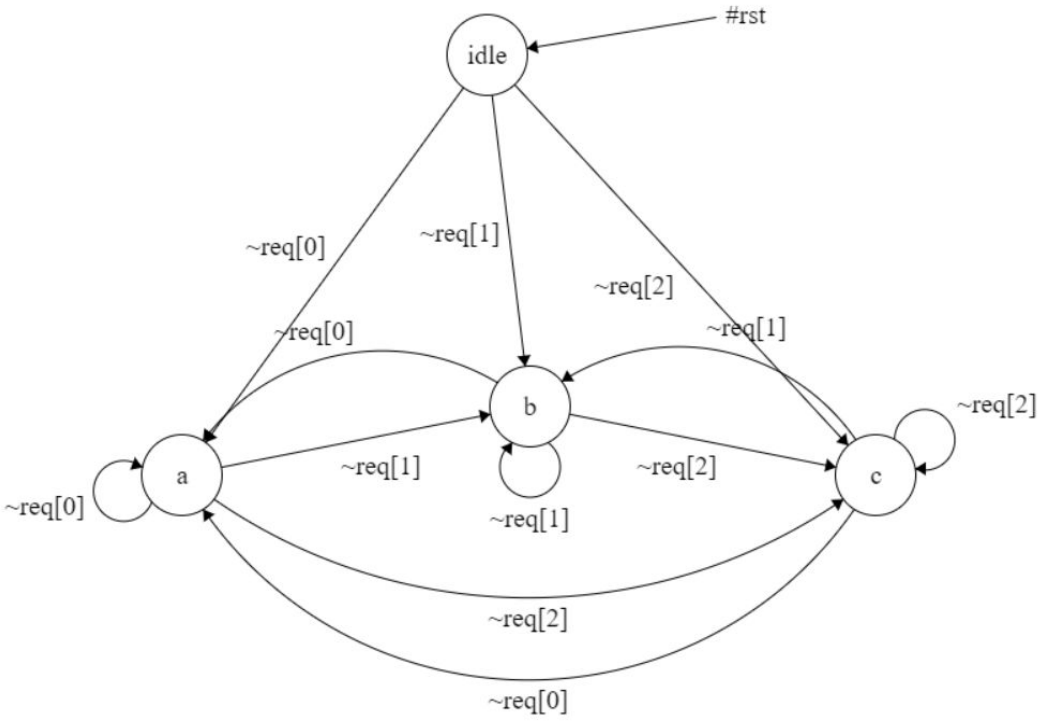

Fig 2. PCI Arbiter Finite State Diagram

D. Verilog Design and Testbench

Below is the Verilog hardware design to implement a PCI Arbiter. The total length of the code is 147 lines, and the test bench is 41 lines. This code came about through weeks of hard work. The final iteration of this code is a result of consultation from Dr. Pang. The following code screenshots include the design of our arbiter. We also made sure that we make the arbiter to be clocked to the same signal.

PCI Arbiter Verilog Design:

```
1  `timescale 1ns / 1ps
2  //////////////////////////////////////////////////////////////////////////////////
3  // Company:
4  // Engineer:
5  // Authors: Miguel Tirado, Sam Lee, Jesus Nunez, Mujeeb , Angelica Smith-Evans
6  // Create Date: 05/06/2019 02:52:01 PM
7  // Design Name:
8  // Module Name: arbiter
9  // Project Name:
10 // Target Devices:
11 // Tool Versions:
12 // Description:
13 //
14 // Dependencies:
15 //
16 // Revision:
17 // Revision 0.01 - File Created
18 // Additional Comments:
19 //
20 //////////////////////////////////////////////////////////////////////////////////
21
22
23 module arbiter(req, frame, irdy, clk, rst, gnt, cnt, trig );
24     //----------------------Inputs/Outputs/Register/constants START----------------
25     input [2:0] req;
26     input frame, irdy, clk, rst;
27
28     output reg [2:0] gnt;
29     output reg [4:0] cnt;
30     output reg trig = 0;
31
32     reg [1:0] cs,ns;
33
34     parameter idle = 2'b00, a = 2'b01, b = 2'b10, c = 2'b11;
```

Fig 3. PCI Arbiter Verilog Code part 1

```
35      //--------------------------------Inputs/Outputs/Register/constants END---
36      //--------------------------------FSM code start----------------------------
37      always @(posedge clk or posedge rst) begin
38          if(rst)
39              cs <= idle;
40          else
41              cs <= ns;
42      end
43
44      always @(posedge clk or cs or posedge rst) begin
45          case(cs)
46              idle: begin
47                        gnt = 3'b111;
48                        if(~req[0]) begin
49                            cnt <= 1;
50                            ns <= a;
51                        end
52                        else if(~req[1]) begin
53                            cnt <= 1;
54                            ns <= b;
55                        end
56                        else if(~req[2]) begin
57                            cnt <= 1;
58                            ns <= c;
59                        end
60                        else ns <= idle;
61                    end
62                a: begin
63                        gnt = 3'b110;
64                        if(!frame && !trig) begin
65                            cnt <= cnt - 1;
66                            trig = 1;
67                        end
```

Fig 4. PCI Arbiter Verilog Code part 2

```
68                        if ((frame && cnt ==16) || (trig && cnt == 8) || (irdy && trig && frame)) begin
69                            trig = 0;
70                            cnt = 0;
71                            if(~req[1]) begin
72                                ns <= b;
73                            end
74                            else if(~req[2])begin
75                                ns <= c;
76                            end
77                            else if(~req[0]) begin
78                                ns <= a;
79                            end
80                            else begin
81                                ns <= idle;
82                            end
83                        end
84                        else begin
85                            ns <= a;
86                            ns <= cnt + 1;
87                        end
88                    end
89                b: begin
90                        gnt = 3'b101;
91                        if(!frame && !trig) begin
92                            cnt <= cnt - 1;
93                            trig = 1;
94                        end
```

Fig 5. PCI Arbiter Verilog Code part 3

```
95                      if((frame && cnt == 16) || (trig && cnt == 8) || (irdy && trig && frame)) begin
96                          trig = 0;
97                          cnt = 0;
98                          if(~req[2]) begin
99                              ns <= c;
100                         end
101                         else if(~req[0]) begin
102                             ns <= a;
103                         end
104                         else if(~req[1]) begin
105                             ns <= b;
106                         end
107                         else begin
108                             ns <= idle;
109                         end
110                     end
111                     else begin
112                         ns <= b;
113                         cnt <= cnt + 1;
114                     end
115                 end
116             c: begin
117                     gnt = 3'b011;
118                     if(!frame && !trig) begin
119                         cnt <= cnt - 1;
120                         trig = 1;
121                     end
```

Fig 6. PCI Arbiter Verilog Code part 4

```
122                     if((frame && cnt == 16) || (trig && cnt == 8) || (irdy && trig && frame)) begin
123                         trig = 0;
124                         cnt = 0;
125                         if(~req[0]) begin
126                             ns <= a;
127                         end
128                         if(~req[1]) begin
129                             ns <= b;
130                         end
131                         if(~req[2]) begin
132                             ns <= c;
133                         end
134                         else begin
135                             ns <= idle;
136                         end
137                     end
138                     else begin
139                         ns <= c;
140                         cnt <= cnt + 1;
141                     end
142                 end
143         default: ns = idle;
144         endcase
145     end
146     //----------------------------------------------FSM code END-------------------------------------
147 endmodule
```

Fig 7. PCI Arbiter Verilog Code part 5

PCI Arbiter Testbench:

```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////
21
22
23  module arbiter_tb;
24      reg clk, rst, frame,irdy;
25      reg [2:0] req;
26
27      wire [2:0] gnt;
28      wire [4:0] cnt;
29      wire trig;
30
31      arbiter m1(req, frame, irdy, clk, rst, gnt, cnt, trig );
32      initial clk = 0;
33      always #10 clk = ~clk;
34
35      initial begin
36          rst = 1; irdy = 0; frame = 0; req = 3'b000;
37          #30 rst = 0;
38          #700 $stop;
39      end
40
41  endmodule
42
```

Fig 8. PCI Arbiter Verilog Testbench

E. Simulation Waveform

Below is the simulation waveform of our test bench:

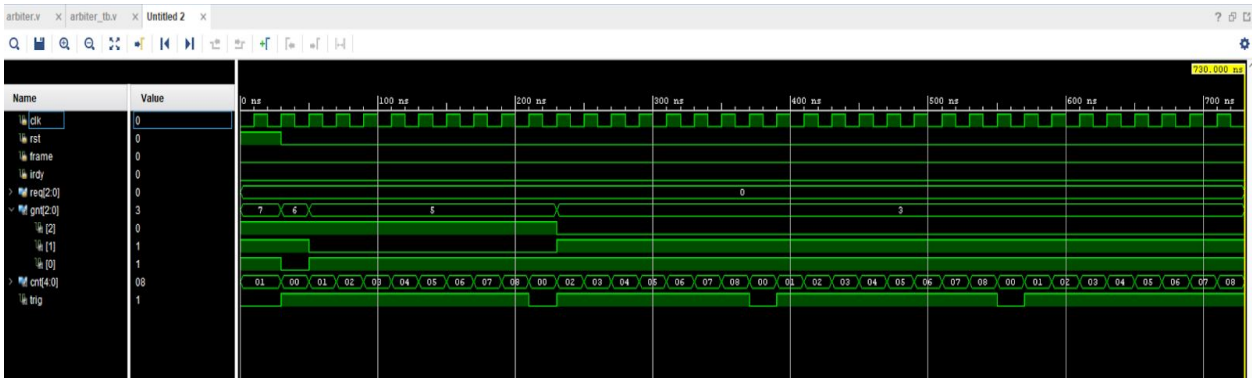

Fig 9. Testbench Waveform

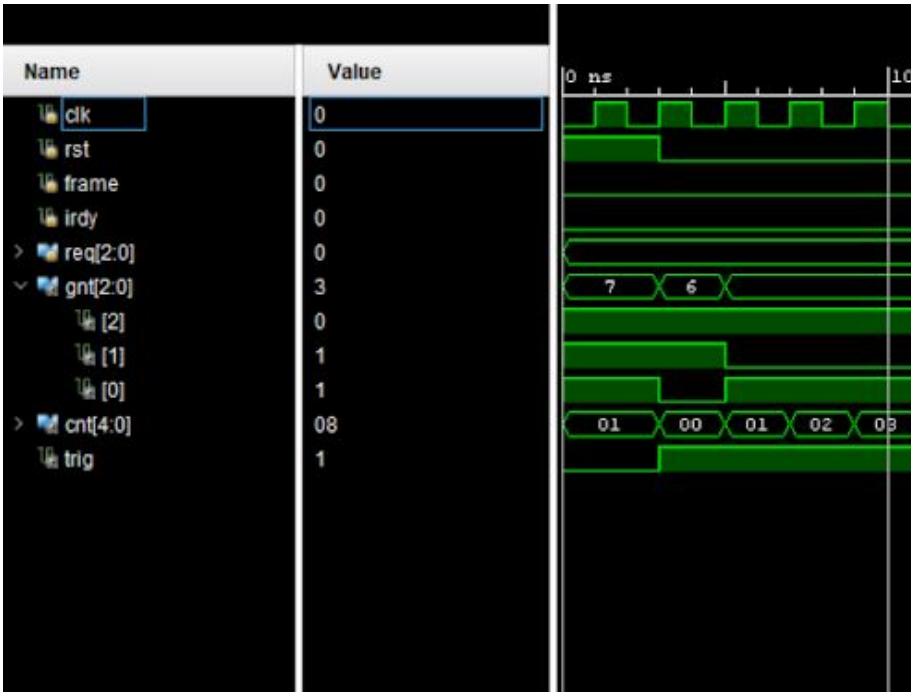

Fig 10. Waveform from 0ns to 100ns

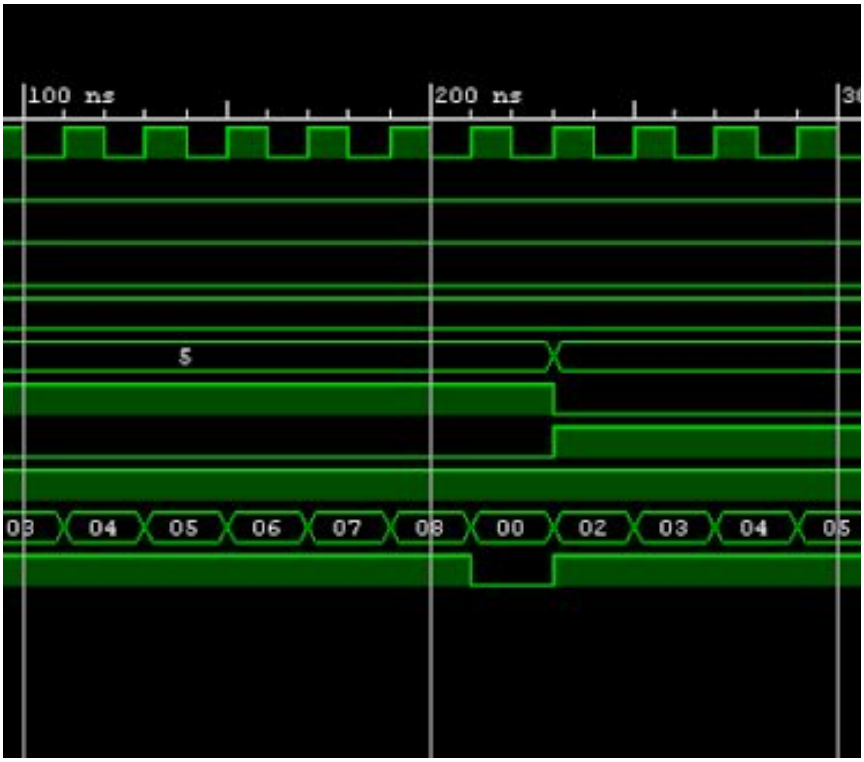

Fig 11. Waveform from 100ns to 300ns

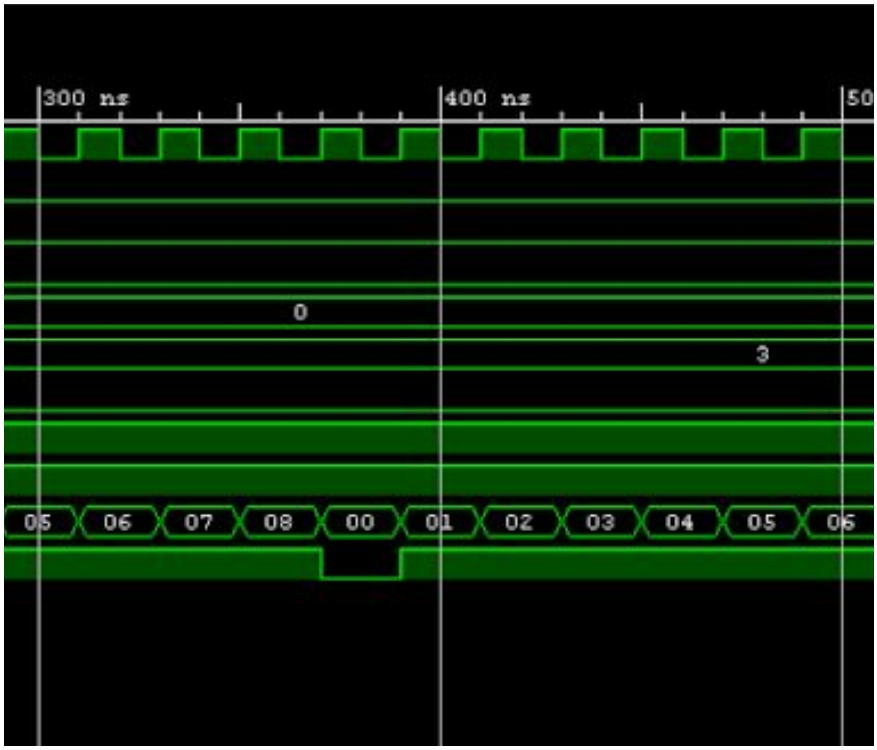

Fig 12. Waveform from 300ns to 500ns

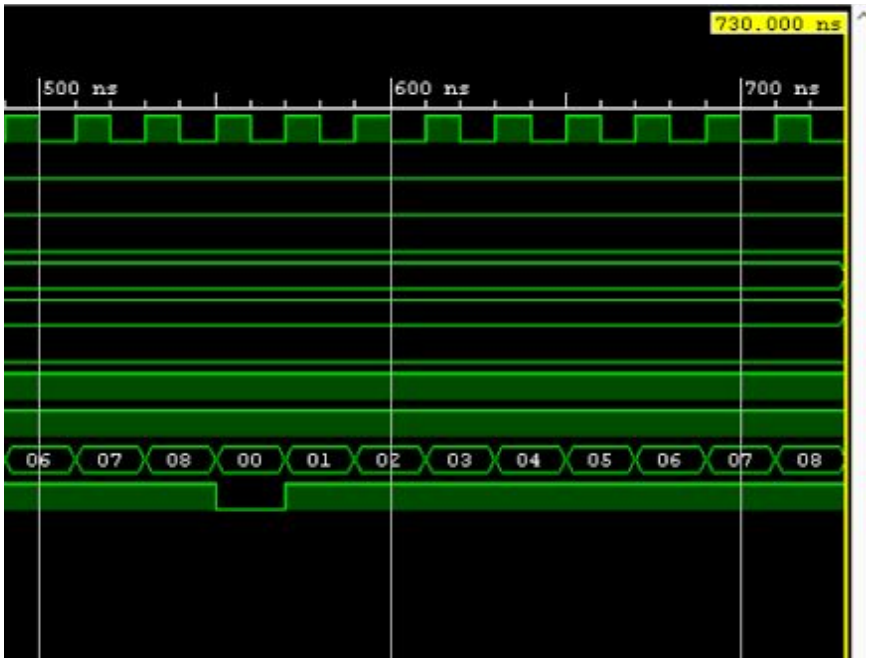

Fig 13. Waveform from 500ns to 750ns

F. Results Discussion:

We started by having the data that we would like to see before we began to start putting the design into code. The results we have achieved shows that the results from the waveform are

matched to our expected results. We tested it by implementing the various type of test cases for the arbiter, so the results of getting an error are minimized.

3. Conclusion

In conclusion, we were able to see how a PCI bus arbiter performs bus arbitration doing the round robin PCI bus arbitration. We also understand how the FSM diagram is supposed to be created for this arbiter design. Once we created the diagram for the state machine and the arbiter schematic we used our knowledge to write the architecture in Verilog for the inputs and outputs. After this operation, we wrote the testbench operation to get a simulated output of our waveforms. This project was very confusing and hard at first because there was no code provided to guide us in it. But Dr. Pang told us to implement a finite state machine and from there we were able to create code based on that. A lot of us struggled with understanding the concept of a PCI arbiter, but through YouTube videos and Dr. Pang's notes on it, we were able to design the PCI arbiter in Verilog and create the waveform for it as well. After consulting with Dr. Pang, we were able to put the finals pieces of the code together and came up with a fully functional simulation of how PCI arbitration works, showing the oscillation of the several signals associated with the arbiter.