**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](#_k2ykc8mwns4q) **3**

[2. PCI Arbiter](#_b2vddlcy9wkl) **3**

[A. Design Purpose](#_yibqh9tbtfgn) 3

[B. Engineering Data](#_m8uru0awz2ra) 3

[C. PCI Arbiter State Diagram](#_lj2qj8lduqj) 4

[D. Verilog Design and Testbench](#_6l9sbanqev7f) 4

[E. Simulation Waveform](#_njn2qpuouneo) 8

[F. Results Discussion](#_qq4itdywpgx2) 10

[3. Conclusion](#_gwj0tn39wos0) 11

# 

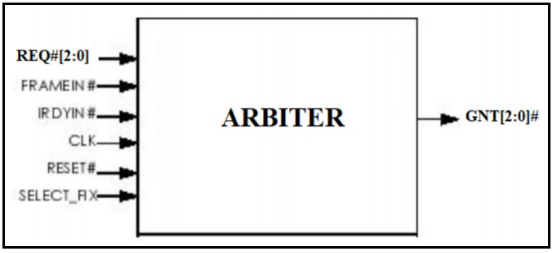
# **1. Introduction**

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:**

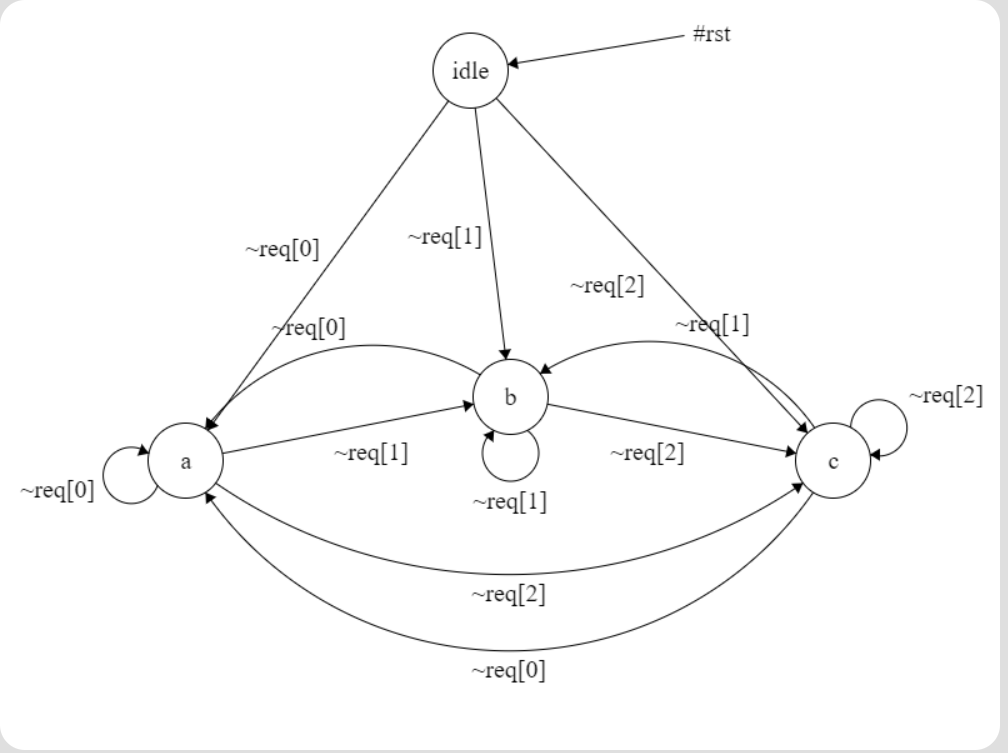
This specific arbiter design shows that we have 6 input signals and one output signal. We decided to design arbiter using a Finite State Machine (FSM) as per the instructions were given to us because it allows us to work with between modules or subsystems. Figure 1 shows the schematic drawing of a PCI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

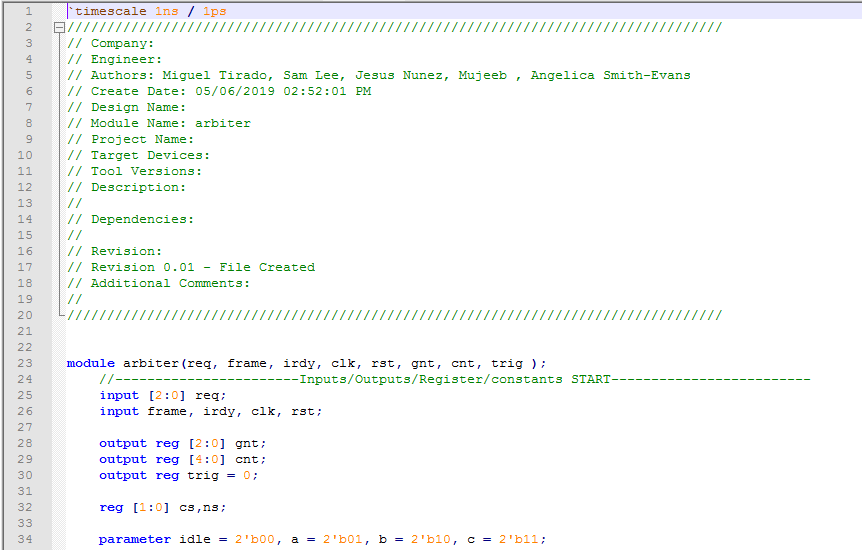


*Fig 2. PCI Arbiter Finite State Diagram*

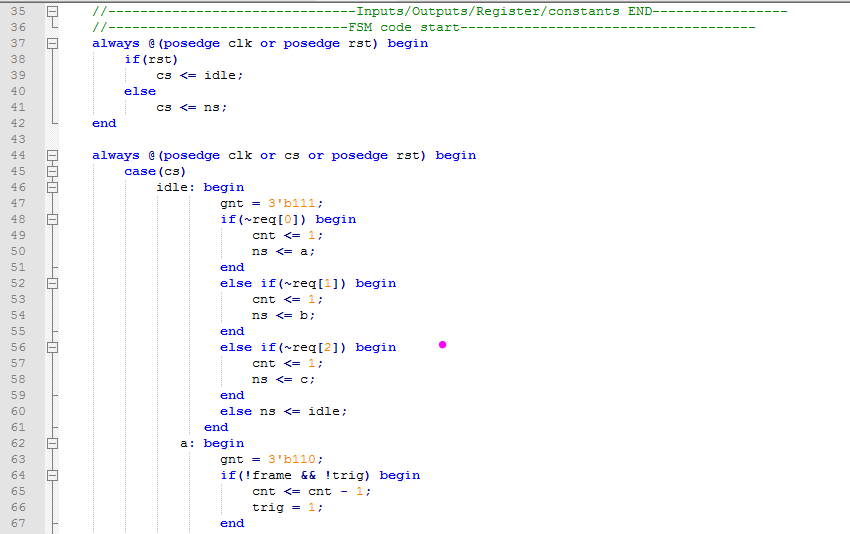
## 

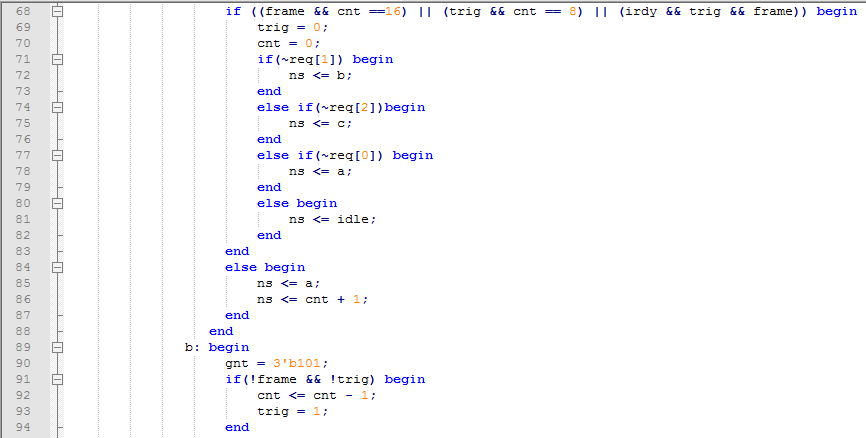
## **D. Verilog Design and Testbench**

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.

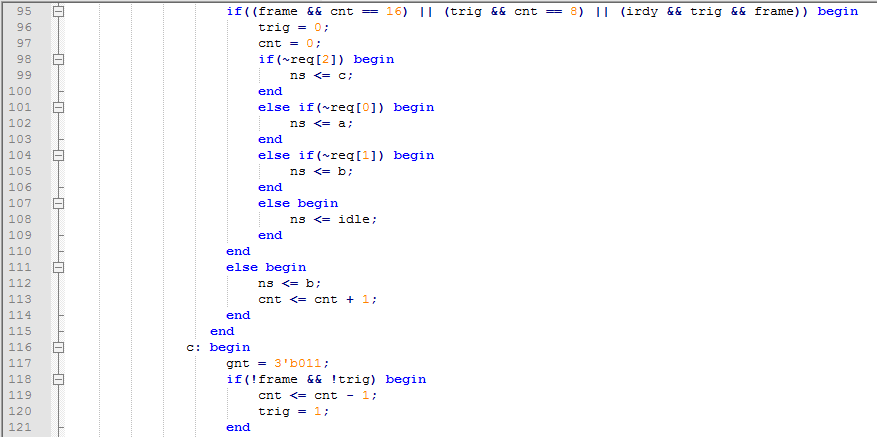
PCI Arbiter Verilog Design:

*Fig 3. PCI Arbiter Verilog Code part 1*

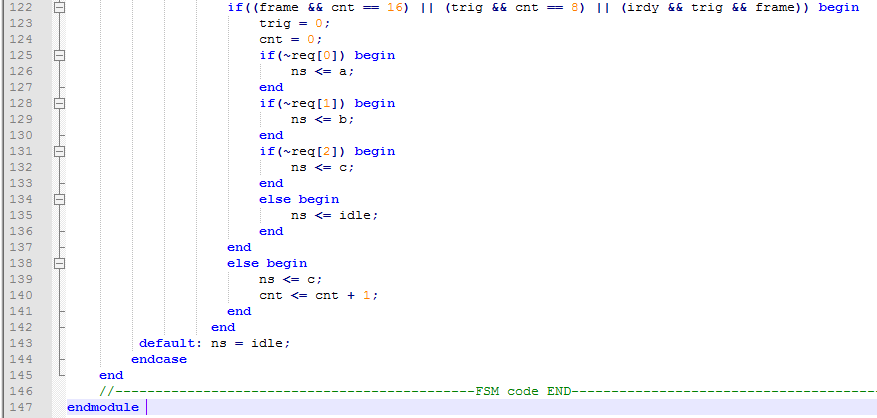
  
*Fig 4. PCI Arbiter Verilog Code part 2*



*Fig 5. PCI Arbiter Verilog Code part 3*

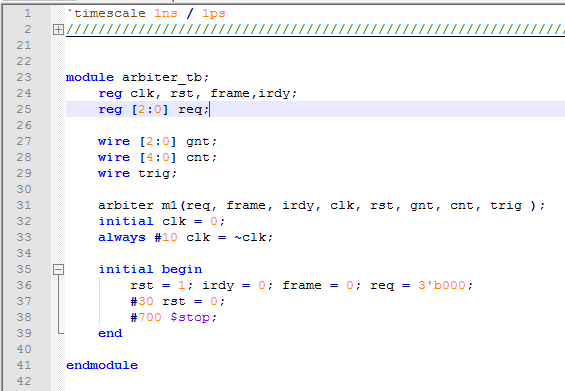


*Fig 6. PCI Arbiter Verilog Code part 4*



*Fig 7. PCI Arbiter Verilog Code part 5*

PCI Arbiter Testbench:



*Fig 8. PCI Arbiter Verilog Testbench*

## 

## **E. Simulation Waveform**

Below is the simulation waveform of our test bench:

## 

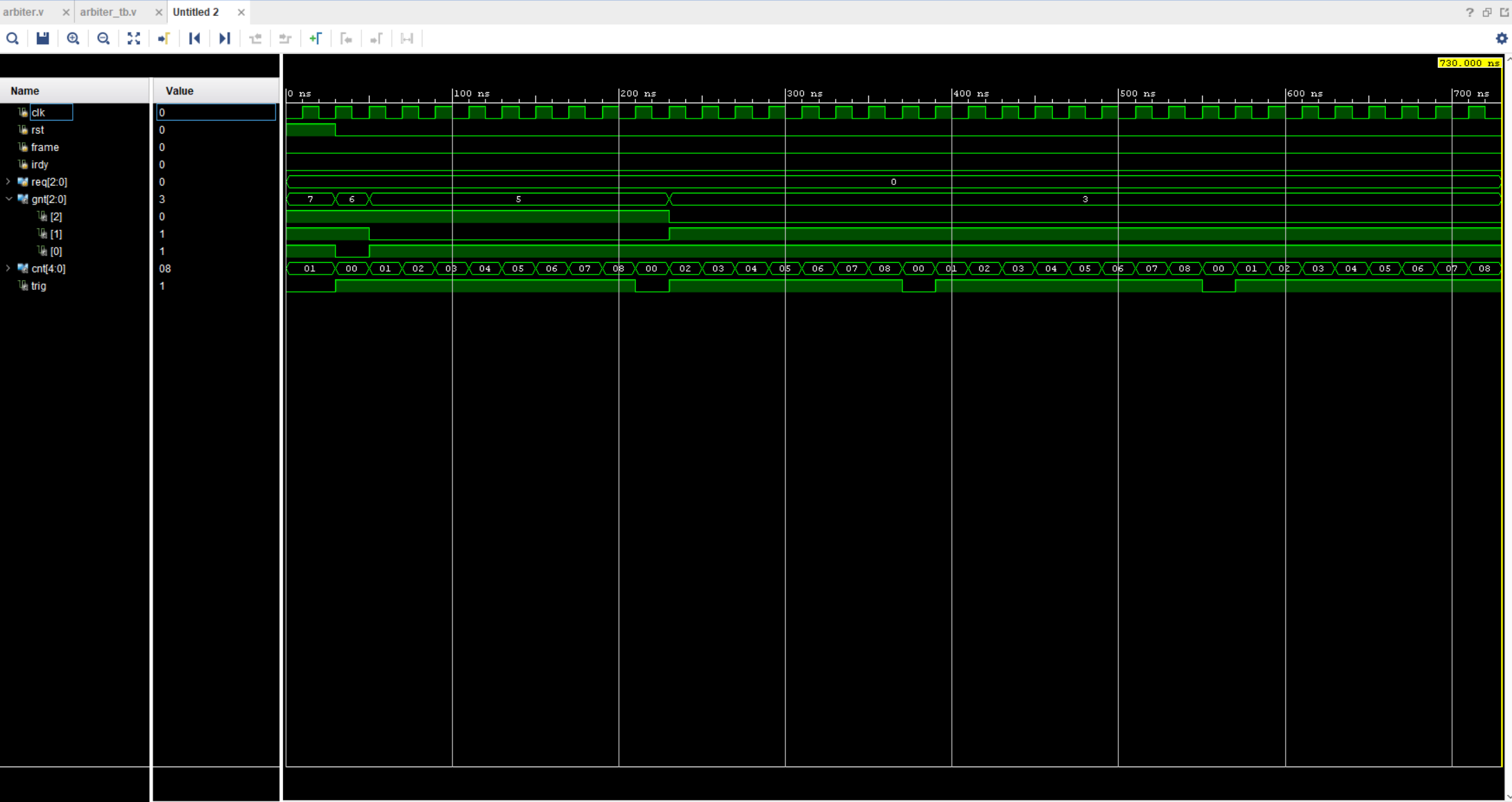
*Fig 9. Testbench Waveform*

## 

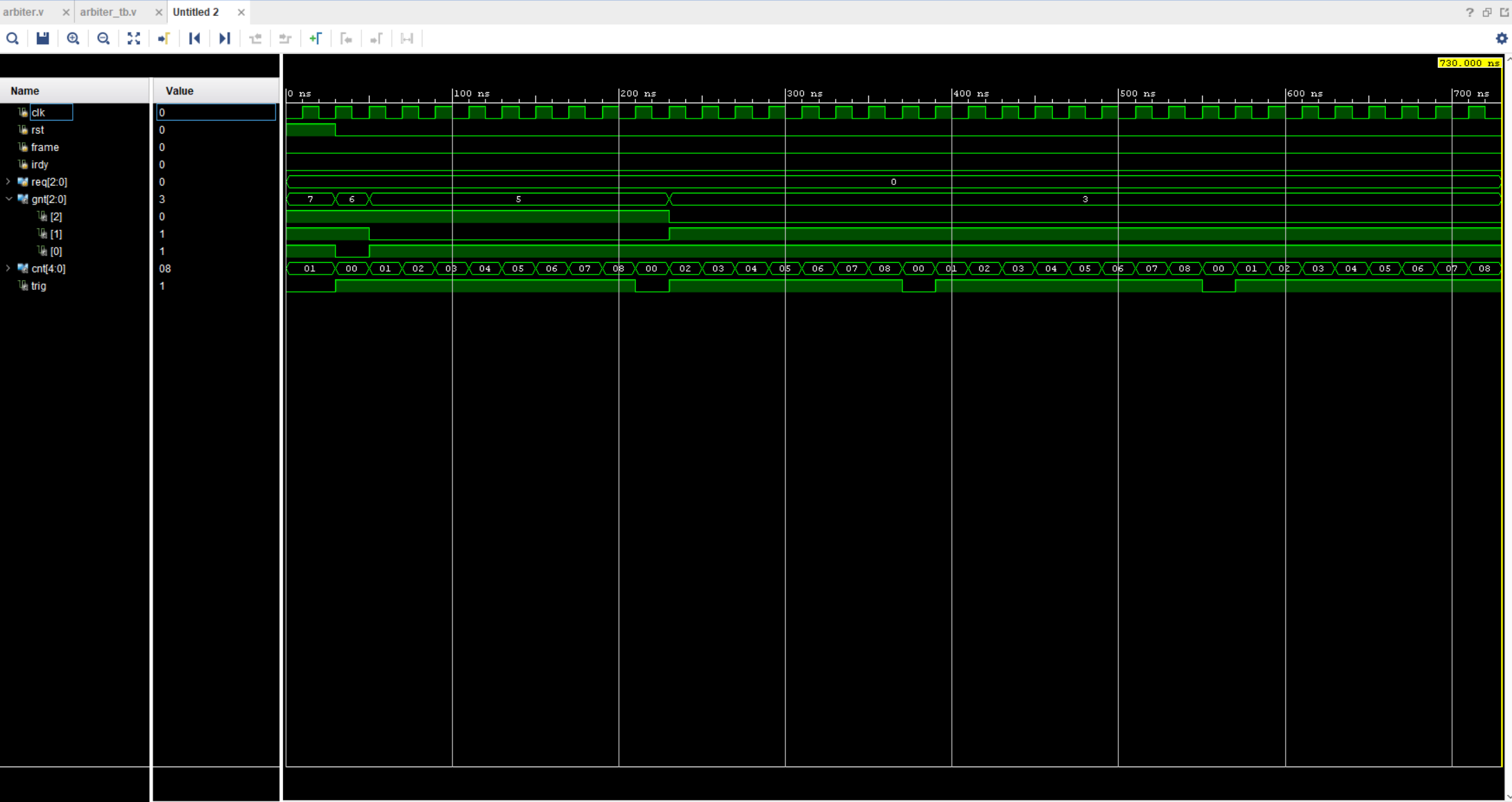
*Fig 10. Waveform from 0ns to 100ns*

## 

*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.