ENEE459D

Final Project

Name: Abbas Paryavi and Allen Onyegbado

Due Date: December 20, 2022

# Introduction

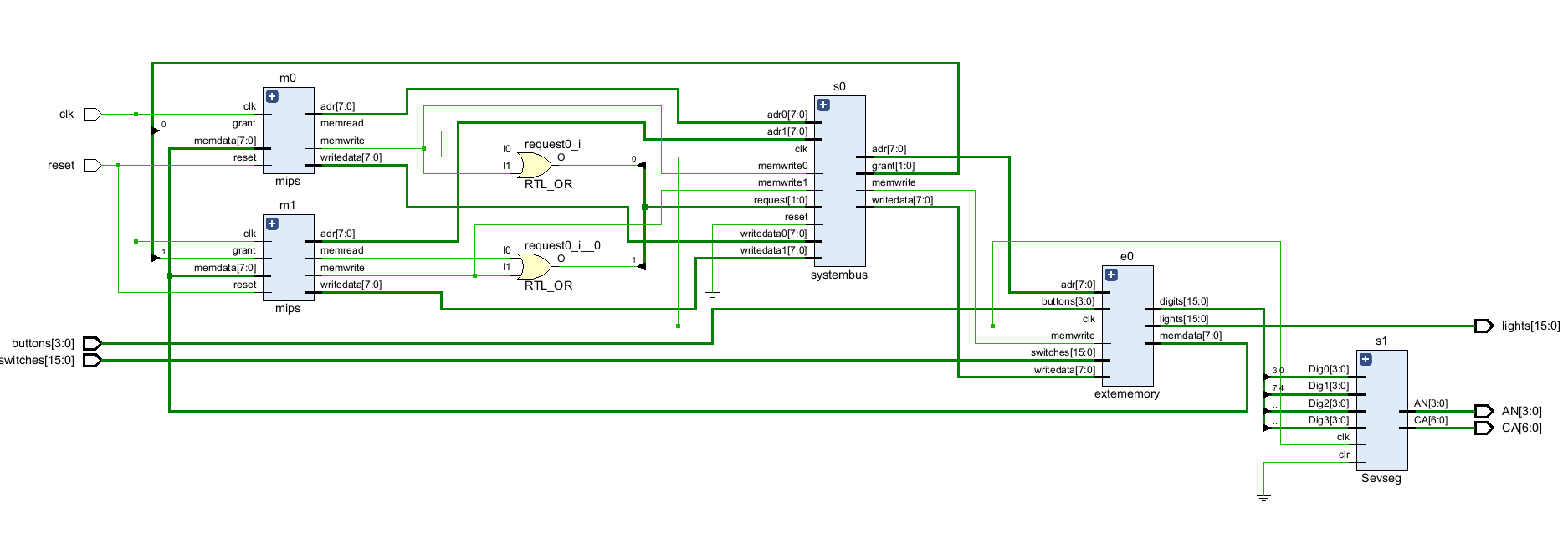
This project aimed at combining everything we have learned thus far in this course and implementing it. The instructions were comparatively open-ended, leaving room for creativity in implementation. There was no structured procedure to follow and a lot of the stuff that was done was completely up to choice. This required a deep understanding of the objective in order to complete it. The purpose of this project was to create a simple MIPS processor and implement it on an FPGA board. However, the objective was also to implement a double-core CPU that accomplishes different tasks with just a switch of a button on the FPGA board. In order to fully test the processors, we were required to have a testbench which we heavily relied on for testing every single scenario so that once we would implement it on the FPGA board, we could troubleshoot the issues with much more ease. For testing purposes, we had to run our processor against a configuration file with a series of assembly-level instructions. Since calculating and putting in these numerous instructions could be a bit tedious, we decided to create a python program that generated the instructions for us. The final task assigned to us for this project was to implement a separate cache for our MIPS processor.

# Hardware and Software Used

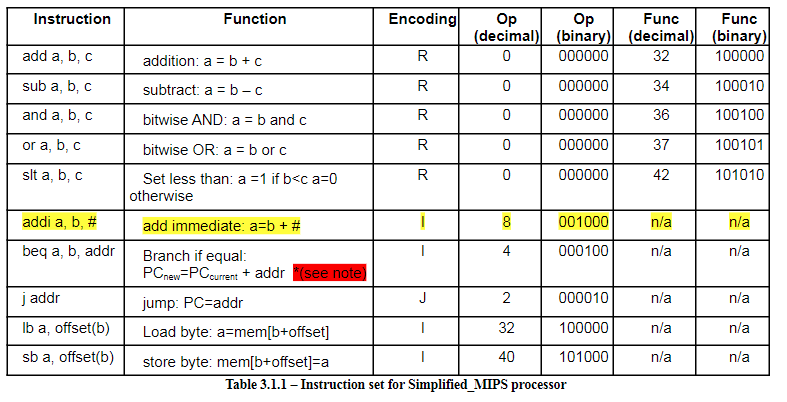
* Vivado
* SystemVerilog
* Verilog
* Python
* FPGA Board

# Architecture

● Block level diagrams –

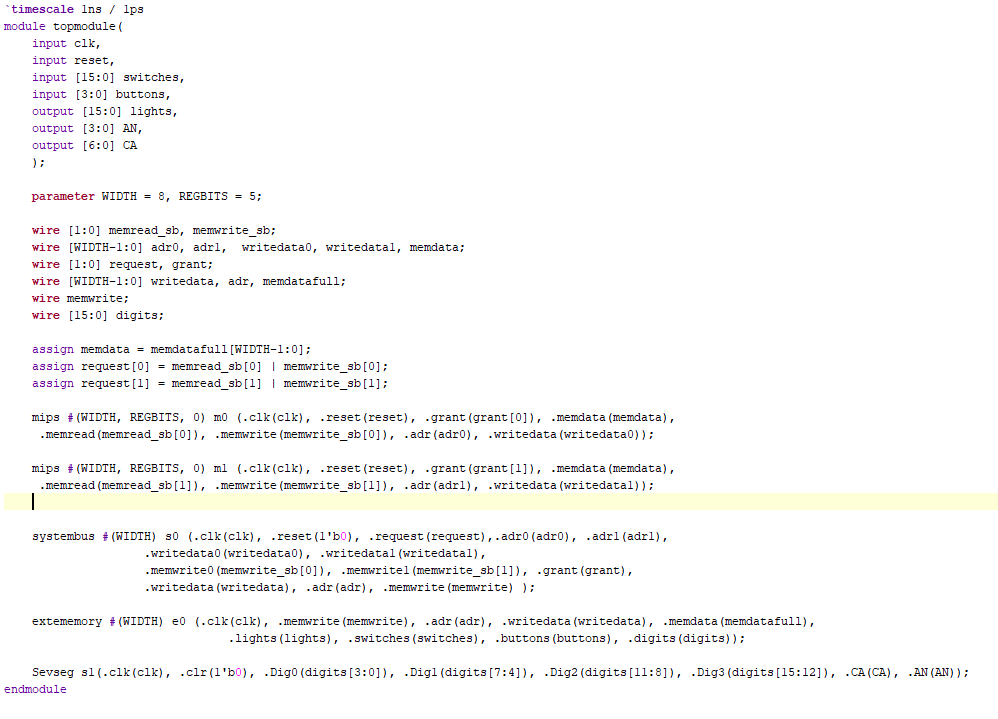


● Assembly instructions the processor implements (a table):  
 Our enhancement was an instruction cache, so the table of instructions implemented is identical to the one given in the final assignment prompt

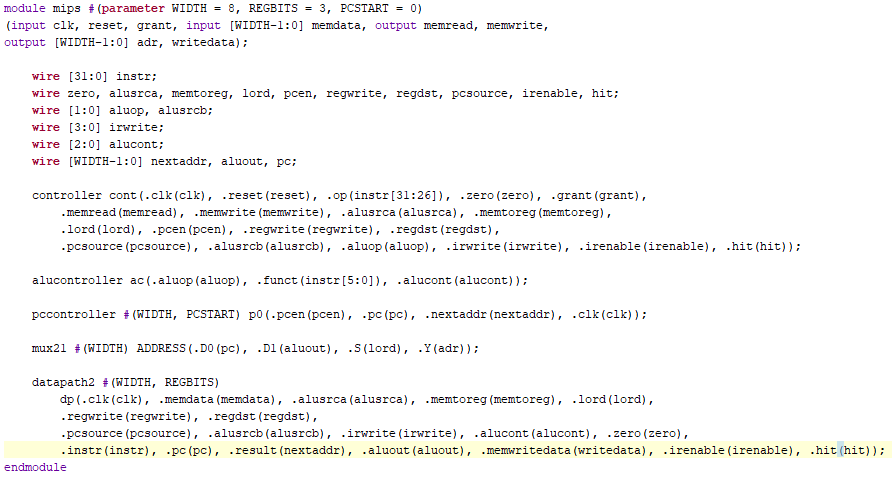


●State diagrams.

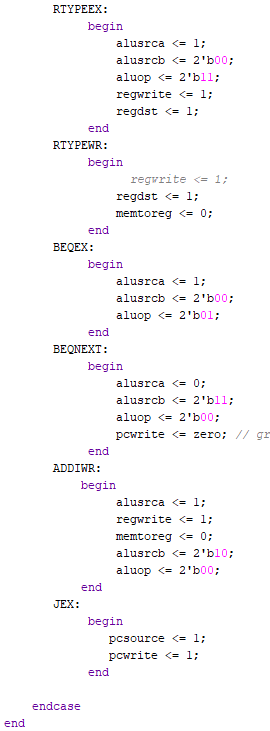
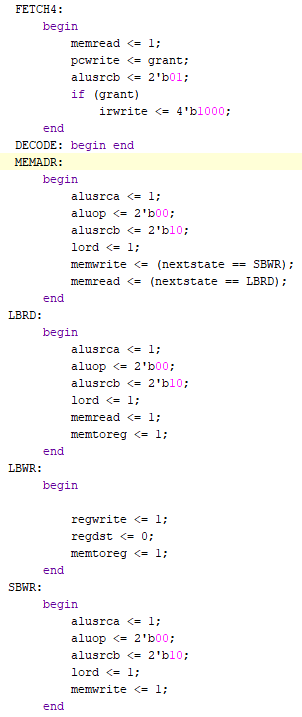
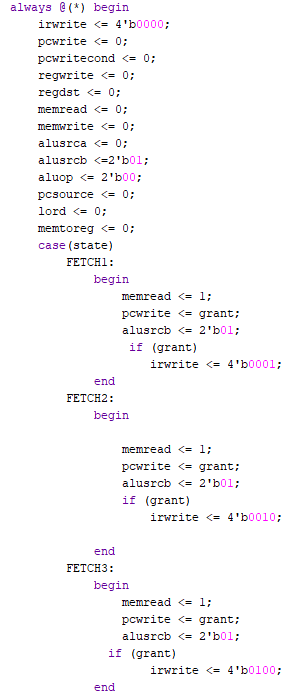
# Relevant HDL Code



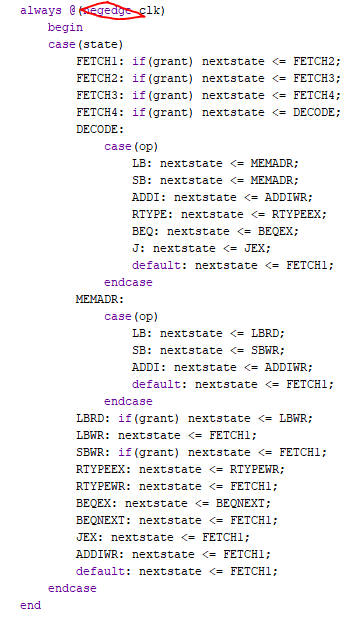
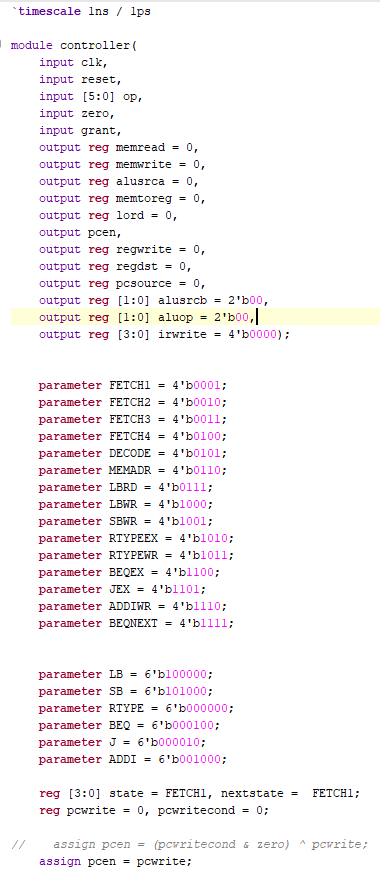
*Top module used to connect all of the submodules.*



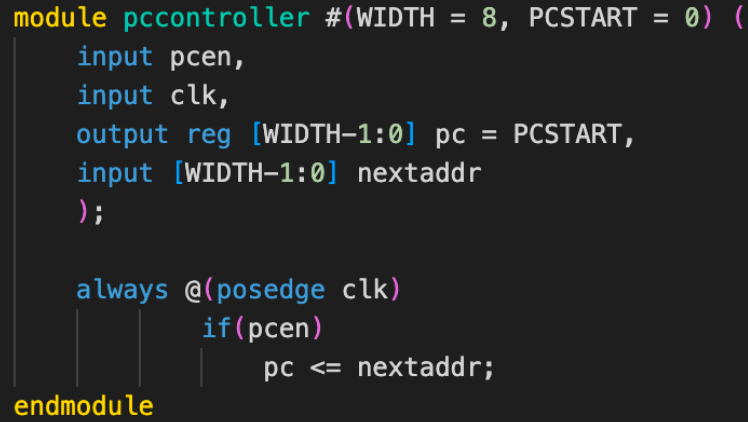
*Mips module that holds the datapath, the controller, and other relevant subsystems.*

**

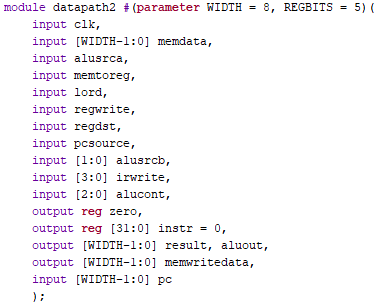
*Controller used to send driving signals to datapath at each FSM state*

**

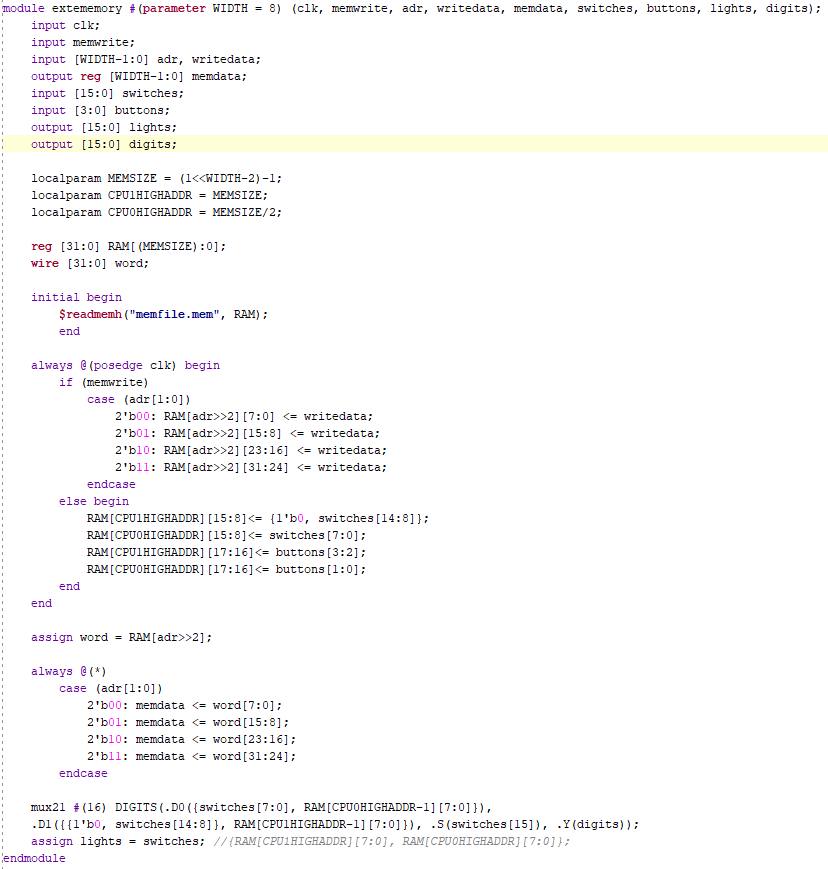
*Controller continued*



*Program counter module*

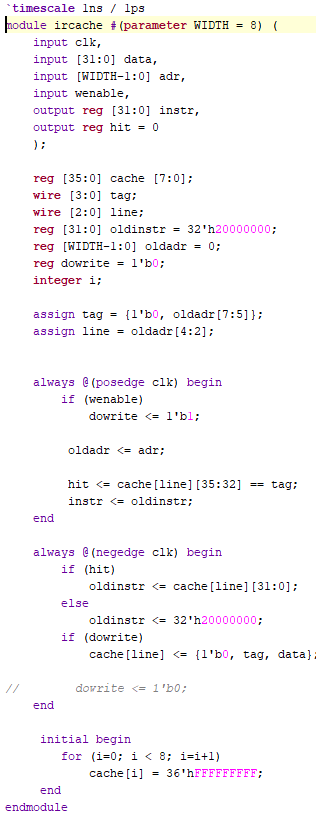
*\*

*Datapath with IRCache*



*Memory/IO*

* 256 bytes total
* 128 per core
* Last 4 bytes used for switch and button input (for each core)
* Next byte used for output to display (for each core)



*Instruction Cache*

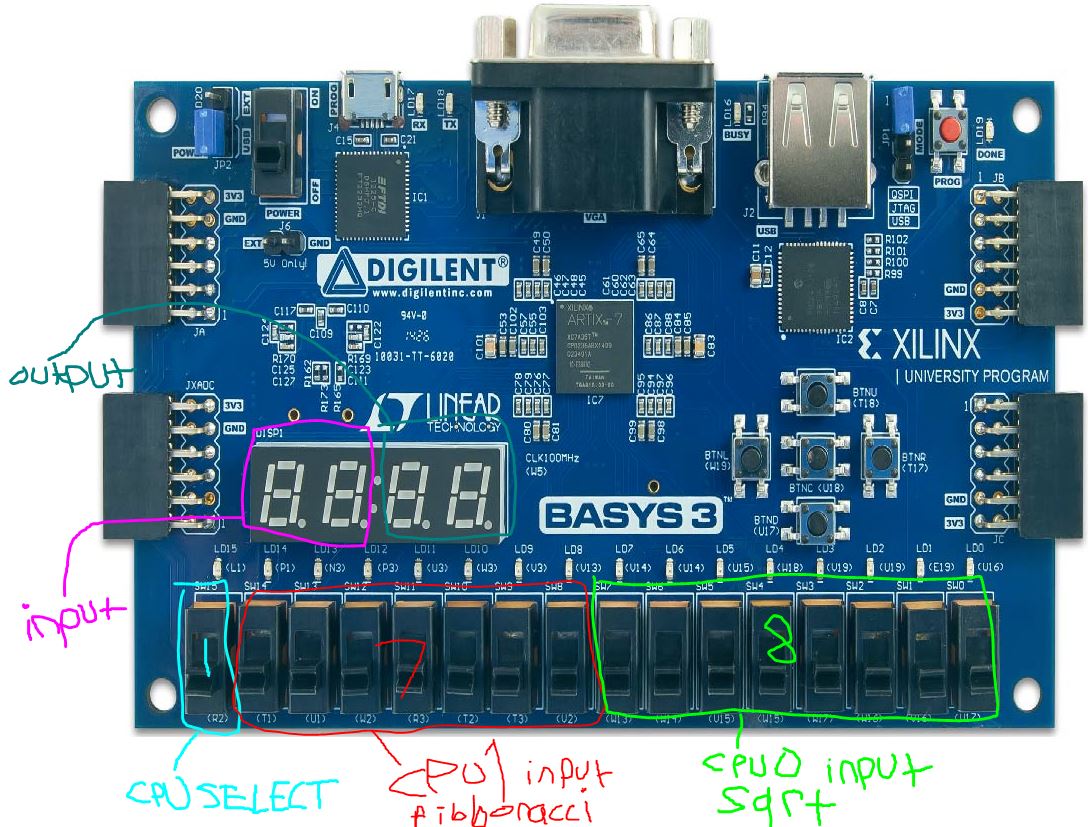
# Synthesis

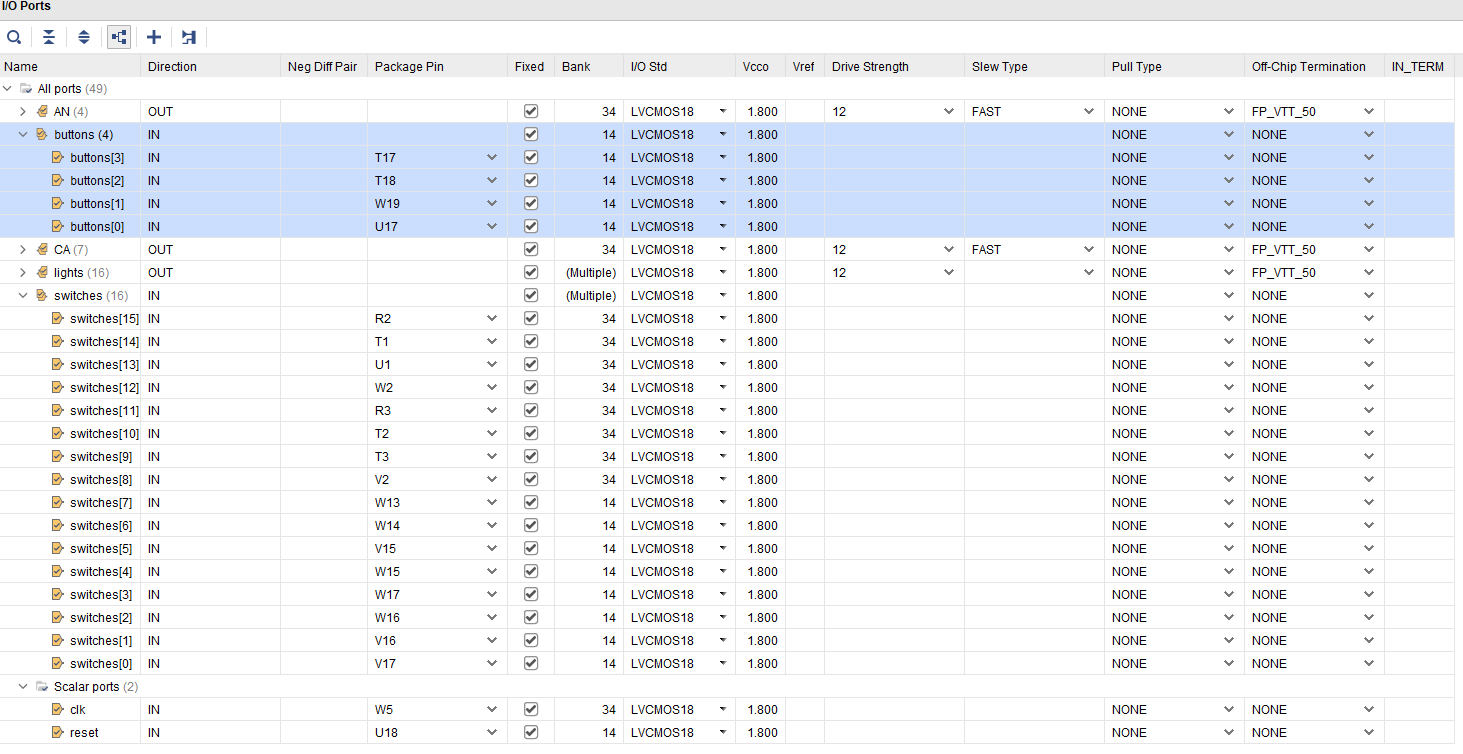
To synthesize the CPU, we had to remove all warnings/critical warnings including inferred latches, and double driven pins.

**Inferred Latches:**

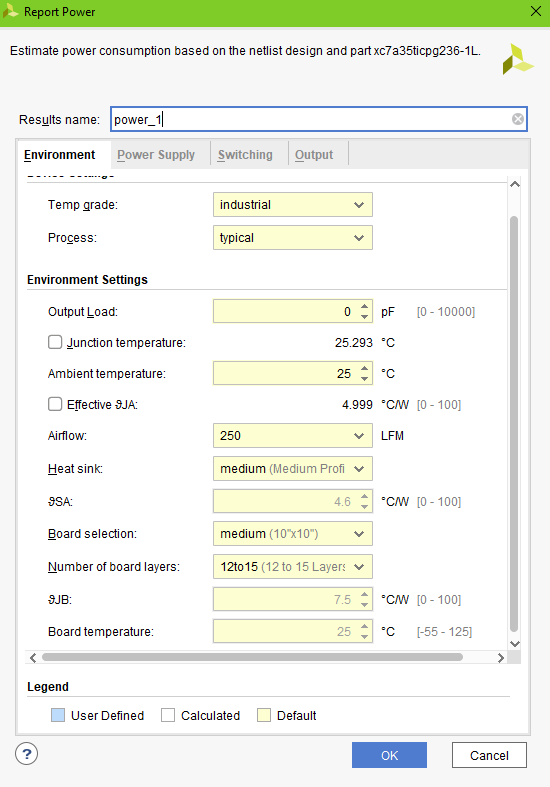
We fixed these by changing case statements did cover all cases or combinational always blocks that needed to be clocked instead of combinational

# Board (I/O) assignment.

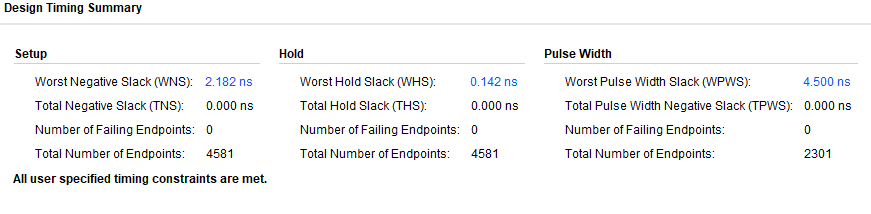




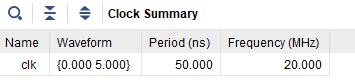
## Synthesis Reports



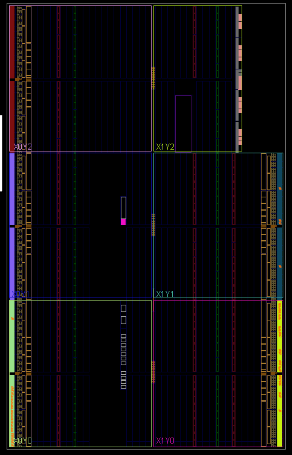
*Power*

**

*Timing*

**

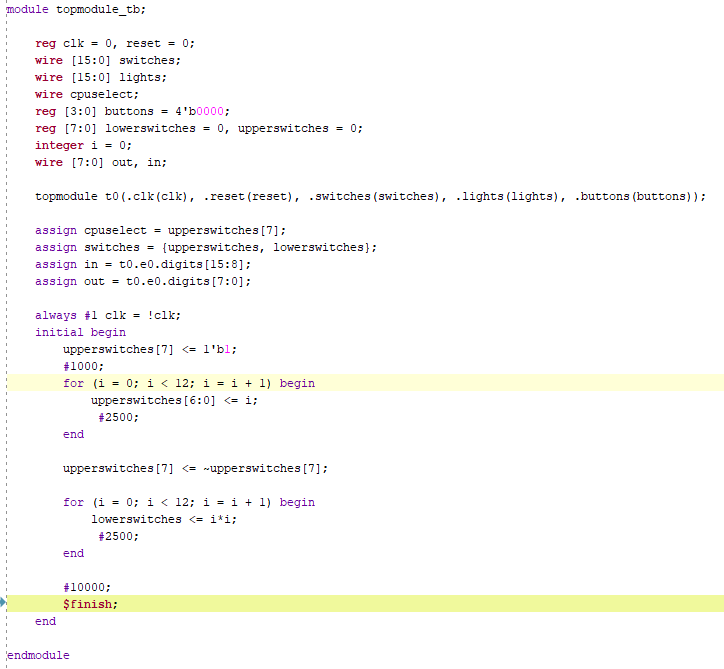
*Clock Summary*



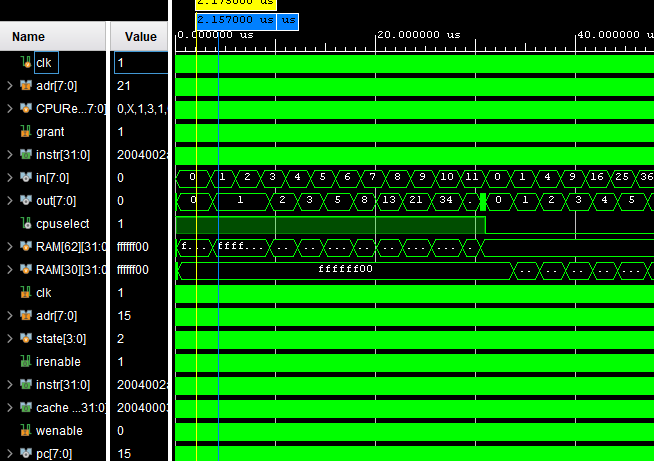
*Area*

# Testing

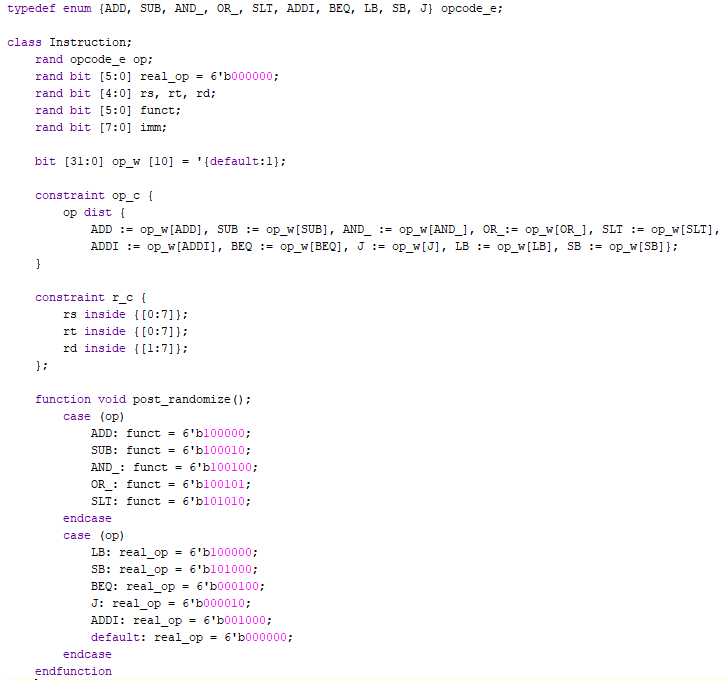
Here is the preliminary stimulus testing we did with themips before adding a formal SV testbench:

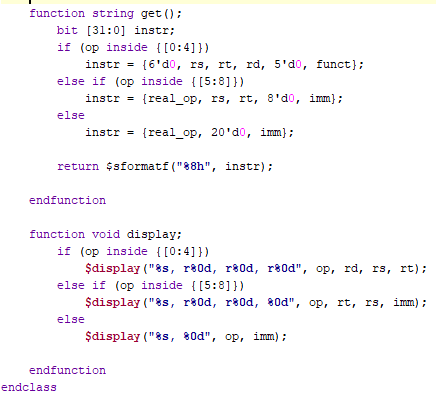


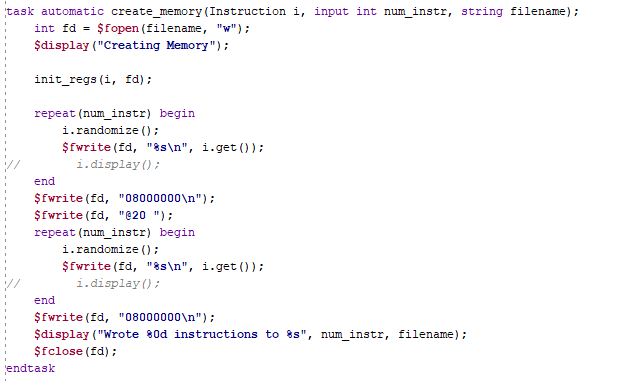
*Informal testbench used for initial stimulus*

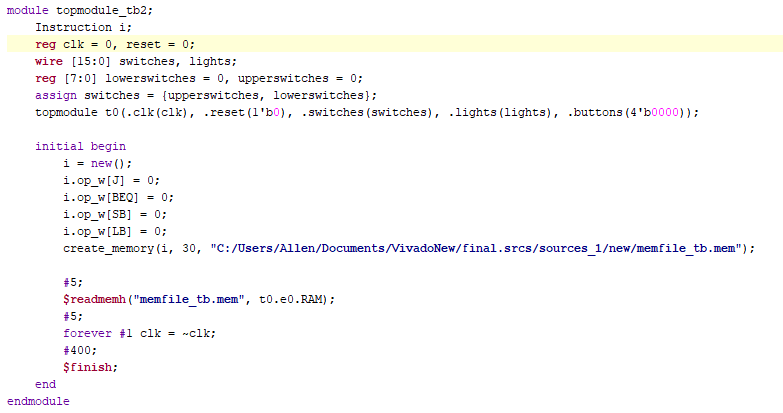


*Simulation from informal testbench*







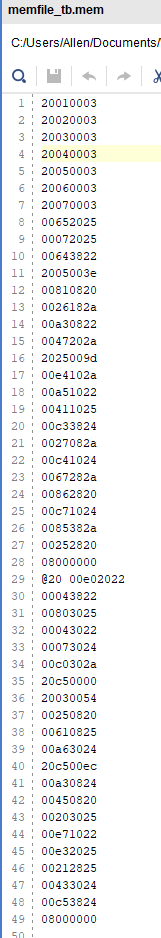


This testbench automatically creates its own constrained randomized instructions to be used on the CPU.

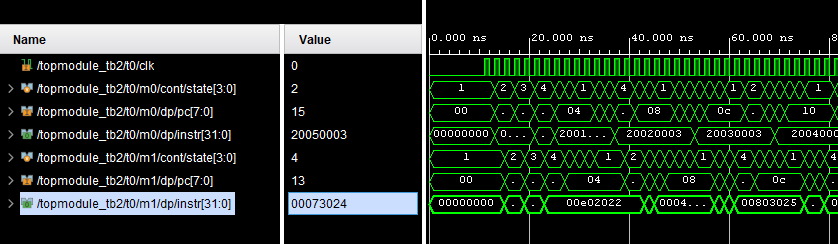
It writes to the randomized instructions excluding jump and branch instruction to a testbench memory file used by the cpu. The weights of the random instructions can be changed dynamically based on coverage.

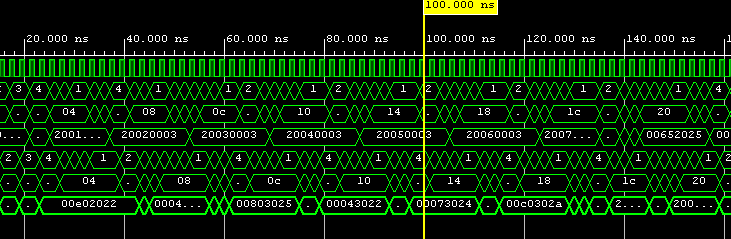


*Weights in testbench for specific instructions modified*



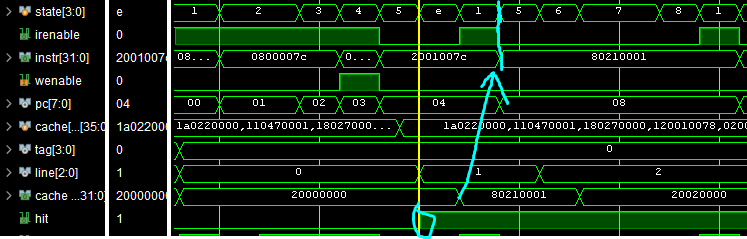
*Generated testbench memory file*

**

**

*Formal testbench simulation*

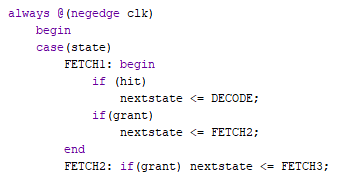
# Enhancements - Instruction Cache



Here you can see the

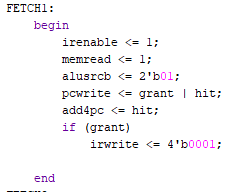
* Multiple parts of the datapath/controller had to be updated in order for the instruction cache to function
* 

This change is in the datapath needs to update the full instruction to the stored cache instruction when a cache hit is detected

* 

In the controller, the state needs to skip the 3 fetch stages and go right to decoding on cache hit

* 



To make up for the missed additions by one from the fetch stages, a new mux was needed with a hardcoded 4 input to add to the pc when all four fetch instructions were skipped

# Problems/Debugging

**Buffered Output:**

Oftentimes we would need to use the last computed value of the ALU in the next instruction, but by the time the next instruction arrives, the inputs to the ALU change and the output changes suddenly. For example, BEQ needed to know if the last result was zero, but by the time BEQ was decoded, the ALU inputs would change and the outputs would change immediately.

**Inferred Latches:**

We fixed these by changing case statements did cover all cases or combinational always blocks that need to be clocked instead of combinational

**Results**

### Square Root:

| This result illustrates the square root of 3 which is displayed on the left two display digits and the result which is rounded down to 1 on the right two display digits. |
| --- |

| Again the result of the square root of 4 is displayed on the left two display digits and the result is rounded down to 1 on the right two display digits. |
| --- |

| Here we have the square root of 15 displayed on the left two display digits and the result is rounded down to 1 on the right two display digits. |
| --- |

### Fibonacci:

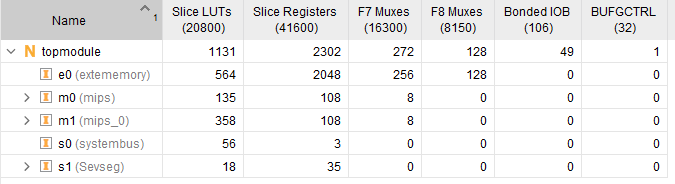
| As can be seen, by the left-most light on the switchboard, we are now on the Fibonacci and we are computing the Fibonacci of 2 displayed on the left two display digits and the result is 1 on the right two display digits. |
| --- |

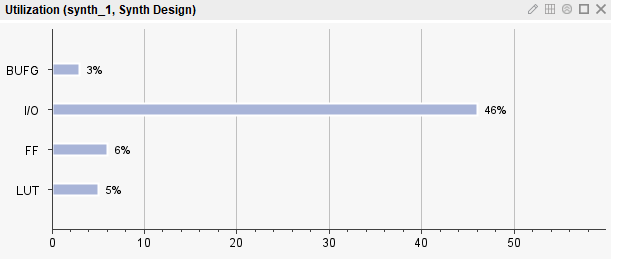
| As can be seen, by the left-most light on the switchboard, we are now on the Fibonacci and we are computing the Fibonacci of 3 displayed on the left two display digits and the result is 2 on the right two display digits. |
| --- |

| As can be seen, by the left-most light on the switchboard, we are now on the Fibonacci and we are computing the Fibonacci of 4 displayed on the left two display digits and the result is 3 on the right two display digits. |
| --- |

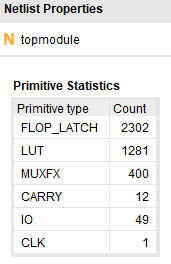
**Conclusion**

**Utilization:**

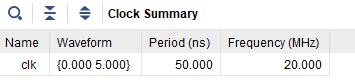
****

****

**Final Pin Count:**

****

**Processor Speed:** 20 MHz



This project was way more comprehensive than any of the previous labs or projects that we had done including the midterm. The instructions were also open-ended thus leaving some room for us to decide what to do to get the project done. For the following reasons, we were given a lot more time to complete this project than the regular weekly labs. The amount of time given to us was reasonable considering the number of issues we faced throughout the project and the level of difficulty it carried with it. One of the first difficulties we faced was actually re-writing the code given to us. We had to go through the lecture video and copy every line of code on the lecture slides onto the appropriate files for the project. This took away a lot of our time however, it was just the beginning of our difficulties. Most of the work we did was done throughout the thanksgiving break and so we worked on it over zoom. Probably the hardest thing about this project was just simply following the diagrams and trying to understand where to connect each module or what modules to even have. However, with a lot of troubleshooting and testing, we were able to resolve many of the issues on the testbench, but it would not synthesize for us to even test it on the FPGA board. It turned out that we had several inferred latches in our design and after fixing those we were finally able to synthesize it and implement it on the FPGA board. But even then, we were getting perfect results for one CPU at a time with the cache attached to it as well, but once we switched it to do the processes of the second CPU we were still having some issues. It took a lot of our time, but we were finally able to have the two processors working simultaneously. Only one of the processor’s outputs would be displayed on the FPGA board based on if the selected switch was on or off.

We faced a lot of challenges doing this project, however, we were able to put together everything we had learned in the span of this course and complete this project successfully without many detailed procedures telling us what to do in every step. This required a lot of time just planning and thinking about what to do, but I feel like this project gave us a more realistic approach to completing projects which are closer to real-world projects, even more than the midterm project. Because we are not always given very clearcut instructions and sometimes have to figure things out on our own.