Erich Viebrock

ECPE 173

Lab #6

Problem Statement

In this lab, implementations are added in order to fix the previously created Beta, as well as add optimizations in order to create a more efficient design. By adding optimizations, our design may be cheaper to manufacture, given we are using less hardware.

Current Design

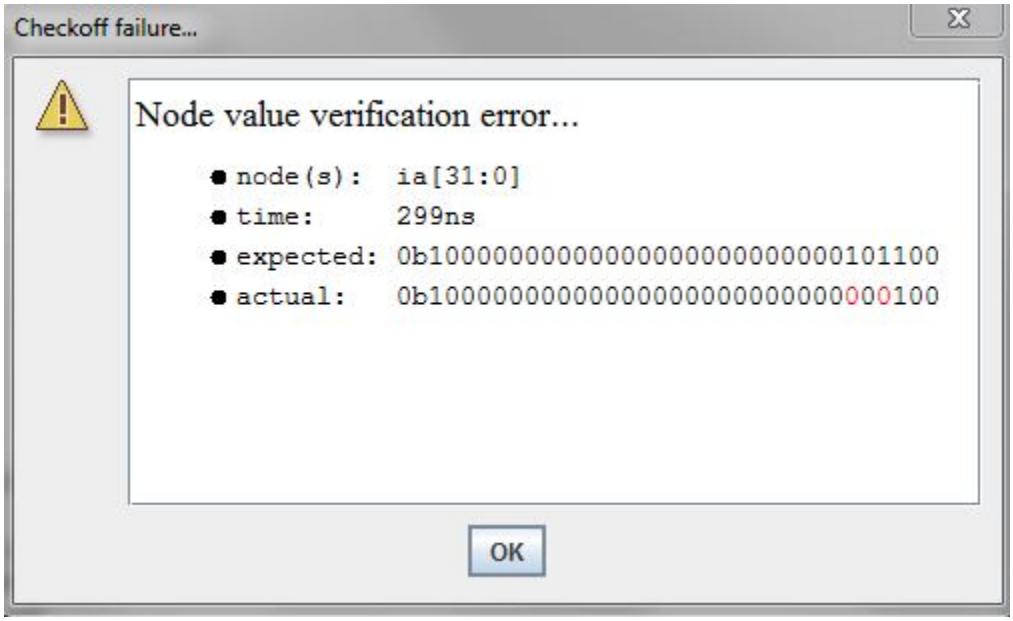
The current design is not complete. There is a large problem of passing variables from subcircuit to subcircuit, as many of the implementations use variables that fall out of scope. Because of this issue, the progress of the design is unknown, being the check off file cannot be tested. The design currently has 2,150 gates, a setup time of 79.518 ns, and a run time of 15.6 μs.

Functional Design

Currently, the design is not functional. However, I have fixed the issue dealing with variable locality, but implementing all of the new logic into the beta subcircuit, as to not corrupt any of the previously created subcircuits. This method works for any additional logic that is added to the previously created Beta, but is not reasonable for logic that has been changed in the new problem statement. For example, the program counter has changed by adding additional functionality to the PCSEL multiplexer.

Therefore, the implementation for this addition must change the original program counter subcircuit.

However, I ran into a significant issue while implementing this step. At 299 ns, the output of the program counter is incorrect.



I am unsure where this issue resides in, because I'm unsure why the output should be the value of expected. After setup, the program counter begins at zero, with the supervisor bit high. After this cycle, we add four to the program counter to move to the next instruction. However, we should be trying to achieve an output of 44.

Optimization

Although I have not reached this section, I have a few ideas to increase speed and cost of the design. First, in order to create some of the variables used in the design, I utilized the buffer operator. This method is a simple method to create variables, as only one line of syntax is needed. However, buffers can be harmful to a design's performance.

By using a buffer, an input is essentially stalled in order to duplicate another variable. Therefore, we must explore other alternatives. There are two instances buffers are used; one instance for mapping constant values to a variable, and one instance for mapping the values of one variable to another variable. Instead of mapping constant values to a variable by means of a buffer, we can just type the variable's values directly instead. This will eliminate the delay the buffer created, causing our design to operate quicker. For mapping variables to other variables, we can use the wire operator in place of the buffer operator. This method is slightly more complicated than buffers, being a dedicated subcircuit must be made in order to create the wire. By using wires instead of buffers, there is no propagation delay in the JSIM environment. However, in the real world, wires have resistance and add a small amount of propagation delay too! These implementations should satisfy our design's need for optimization.

Future Work

Once I have added all the functionality of Lab #5 and Lab #6, the Beta processor should be a fully functional processor. However, there are still additions to add to the Beta processor to make it perform tasks quicker. For example, the Beta processor is currently a single instruction processor. If we added the ability to pipeline our instructions, our processor would be much more efficient. However, this implementation requires precautions for data hazards, as some instructions may need results from previous instructions. In order to handle these hazards, extra control logic must be included to NOP pipeline stages. Once we have implemented a processor capable of pipelining, we can add cache levels to the processor. In order to make our cache even faster, we can also add a page table and TLB in order to make lookup times quicker.

Once we have enabled our processor to utilize caches, we can then move to a multicore processor. In order to implement this concept, minimal changes would need to be made to the processor internally. The bulk of the changes would reside in the processor I/O, in order to allow the processor to interface with other processors simultaneously. Similarly to the data hazards in creating a processor capable of pipelining, a multicore processor can run into the same situation. If one of the processors is reading/writing to a memory address that another processor is reading/writing to, this can propose an error. In order to avoid these issues, control logic is needed to NOP one of the processors over the other. Once we have created a multicore processor with caches and the capability of pipelining, we can then easily implement a spintronics processor.