California State University, Fullerton
Computer Engineering Program

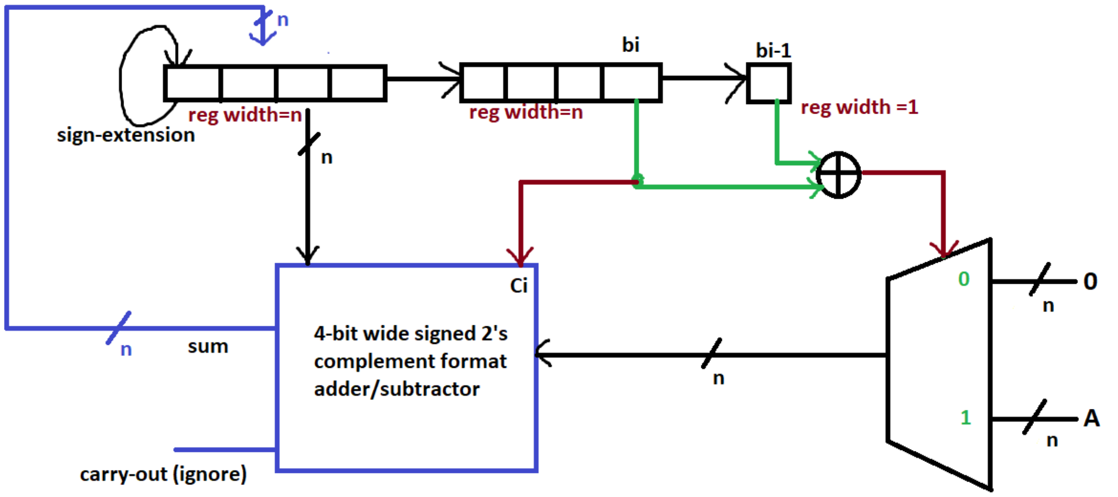
EGCP 446 Advanced Digital Design using Verilog HDL Fall 2020

Exercise #5

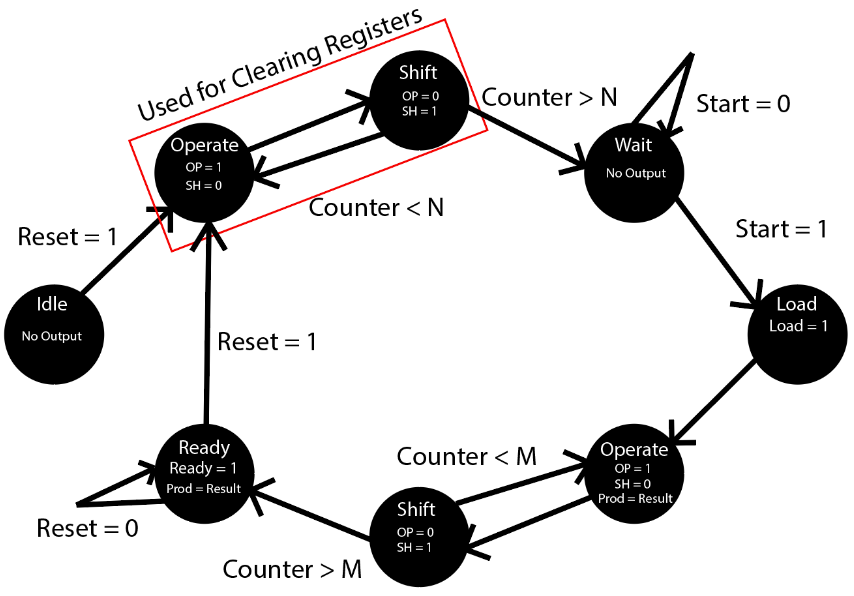
(Team-based)

By Levi Randall and James Samawi

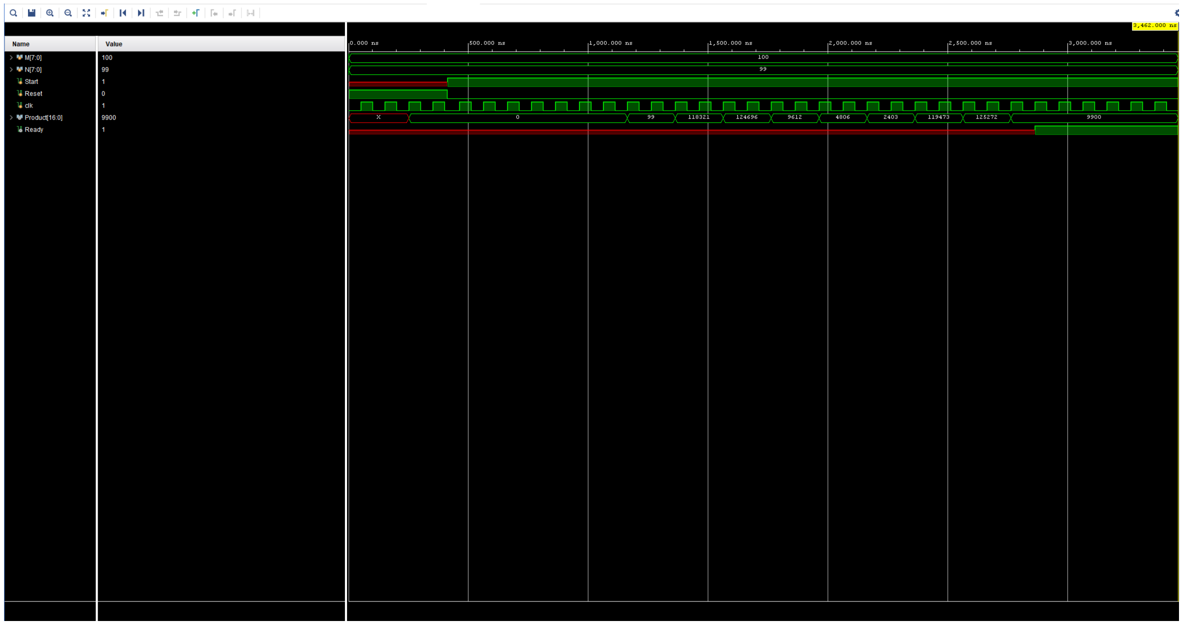
(Due on December 4, 2020)



- 8x8 Booth's Multiplier Circuit Diagram



- Booth's Multiplier FSM and Flowchart



- Sample Waveform Output for A = 100, B = 99.

Problem Statement:

In this exercise, we attempted to create a Boothe's multiplier in Verilog. This multiplier has a very specific compared to other multipliers and due to its conceptual nature, it is being coded on the behavioral level within Verilog. In total, all components were create on the behavioral level which included a 2n+1 bit wide right shifting register, a single bit XOR gate, an 8bit wide 2to1 Multiplexer, an 8 bit CLA Adder/Subtractor, and an external control unit that creates necessary signals for the multiplier to operate.

Methodology:

To begin, we started with the smallest components first and gradually made bigger components. The XOR was first to be coded and here we learned that registers were going to be used during almost everywhere in the project. We then created the MUX and then created the CLA Adder. Due to previous projects and our Term Project, both James and I, have already had experience both with Carry Look Ahead Adders, Adder/Subtractor Circuits, and Multiple Bit Wide 2to1 Multiplexers and thus, finished the beginning without any major issues.

Here we moved wrote test benches for each component to verify their use. We then created a 17bit wide Shift Register and at first had few problems but later ran into many. A new component was made to then house all the components that we labeled as the multiplier. We discovered many issues with the register and realized that the whole circuit almost had a pipeline effective if the enable signals weren't cautiously conducted.

After many problems that will be discussed later, we got that circuit working and created another component of equal level with our multiplier to supply the necessary Operate, Shift, Load, and Ready signals called the control unit. We setup a complicated module that used if statements and a counter to index called "done" to know when our circuit was done at certain intervals. The control unit also supplied these other signals in an alternating fashion to effectively pipeline the signals and would shut off the multiplier when the correct result was achieved. A final component was made to house these two level circuits and took in user inputs for test bench purposes. The code functions effectively and works successfully as a Boothe's Multiplier.

Discussion:

Our results were exactly as expected. The Multiplier would iterate slowly over the numbers we had given and after enough cycles we could watch as the number slowly turned into the answer we were expecting. It also functioned effectively when given a negative number. The negative numbers did originally have a problem where the Most Significant Bit was not being sign extended but this was due to a small variable assignment mistake.

Technical Challenges:

A large majority of this lab's difficulty lies within managing signals as if the component was a multistage computer. The registers gave us a large amount of trouble trying to figure out how to initialize them to have all 0s so that our conditional statements would work appropriately and ended up using a reset followed by a value load in order to flood the system with 0s initially so the circuit could work, otherwise the system would only reflect XXXX values. Additionally, we attempted to not use the shift variable for shifting and found that it was too difficult without it. Our answers without it would be almost perfect but would always be shifted one time off the number it should have been. Other unforeseen problems were the values in each always @ block not updating immediately, meaning multiple if else statements had to be used in conjunction with other if statements instead of simply updating one variable that could be reflected in pre-existing if statements.

Conclusion:

In all, this project was very challenging and was unlike previous exercises. The heavy reliance on registers made the flow of variables react different and made everything react at the pace of the clock. The overall structure of the circuit was also very challenging, learning how the circuit functioned was a good exercise all on its own. Another interesting task was how almost all components were done with behavioral logic which made even simple XOR gates more of an adversary. We both learned much more about Verilog behavioral coding as well as the registers variable type.