ECE 385

Fall 2019

Experiment #4

Introduction to SystemVerilog, FPGA, CAD, and 16-bit Adders

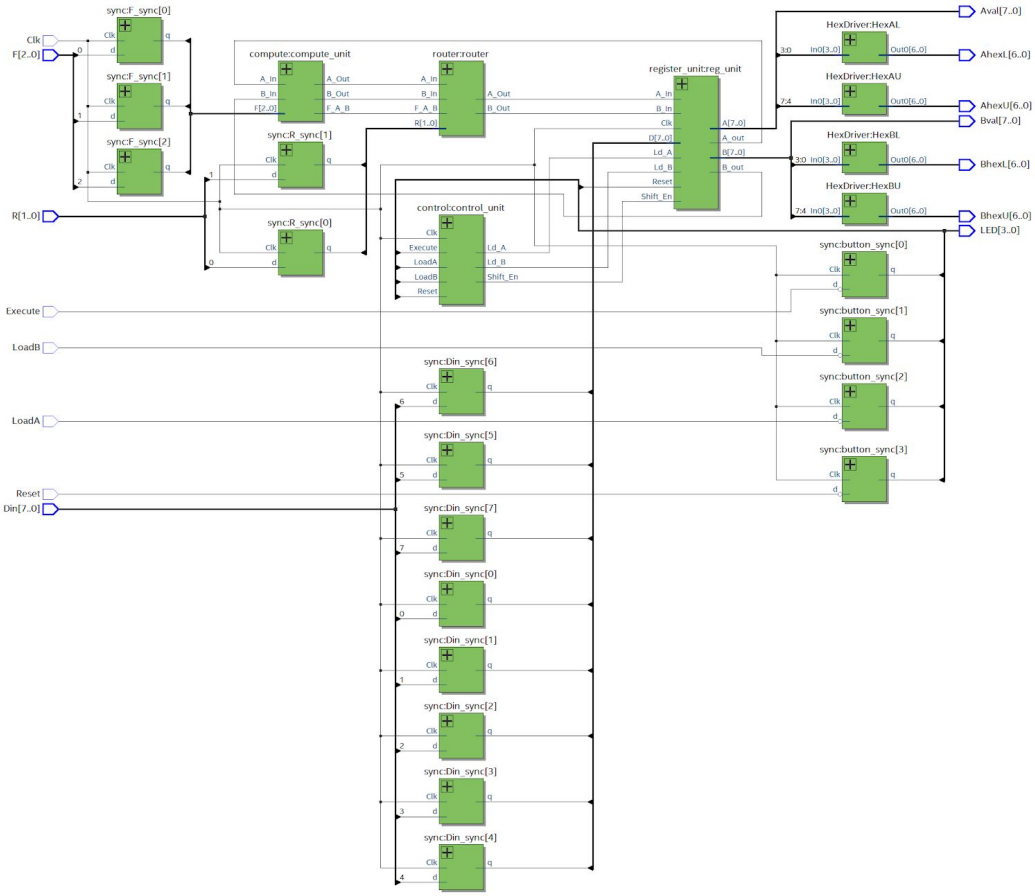
Eric Dong, Yifu Guo
ericd3, yifuguo3
Section AB3, Thursday 2pm
Gene Shiue

1. Introduction

   In this lab we modified a serial bitwise processor as well as constructed 3 different types of adders in SystemVerilog. For the serial processor, we were supposed to extend the given 4 bit processor to an 8 bit version. It should be able to calculate bitwise operations for 8 bit inputs. For the adders, they are designed so that it is able to add together two 16 bit bit strings. We closely examined how to speed up the addition process by looking at the carryout bits.

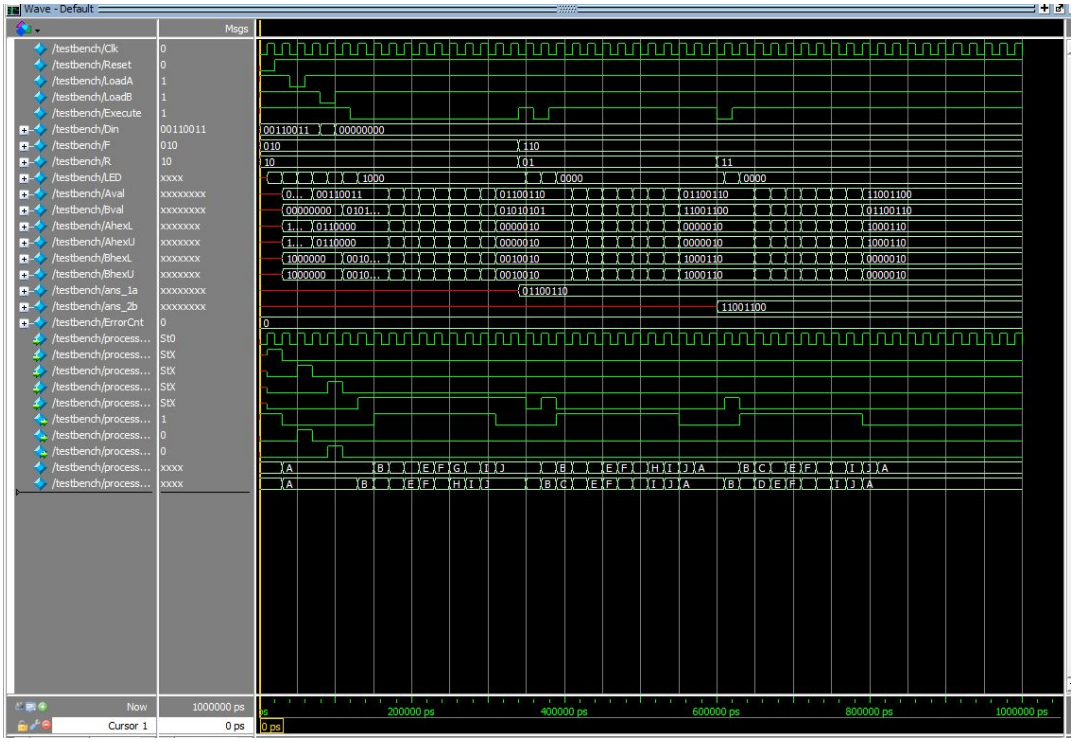
2. Part 1 - Serial Logic Processor

   a.

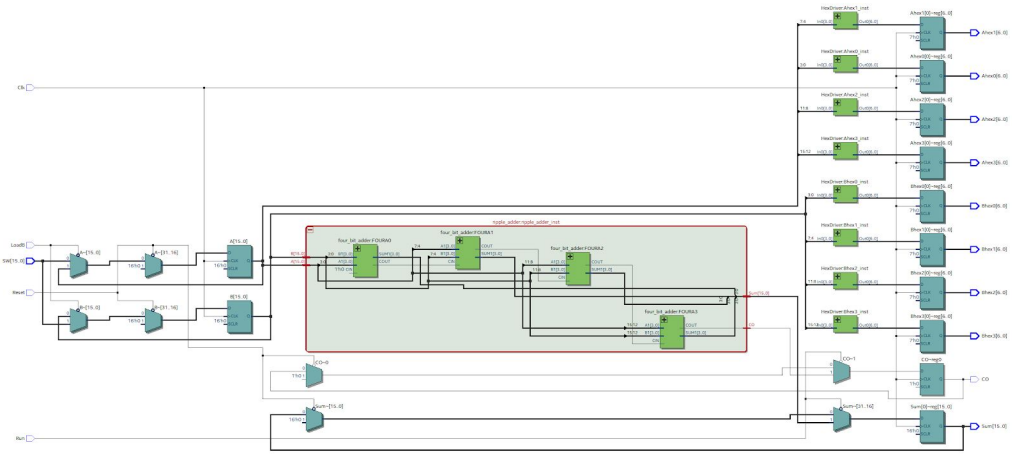


   b. In order to extend the processor from 4 bits to 8 bits we had to change the number of bits that the register can hold, extend the number of states, implement what the next state logic should be and we also had to change the testbench so that it could display the answer on the simulation, lastly we also had to change how many hexdrivers that we had to use.

c. The operation being performed was 0x33 XOR 0x55. And since R has value 10 then the answer should be stored into register A.



3. Part 2 - Adders
   a. Ripple Carry Adder
      i. The ripple carry adder is a series of full adders, in total 16 of them, groups of 4 full adders. Each full adder is serially connected to the next one. The carry out pin of one adder is connected to the next adder's carry in pin. Each full adder takes in 2 bits and adds them up.
      ii. Block Diagram:



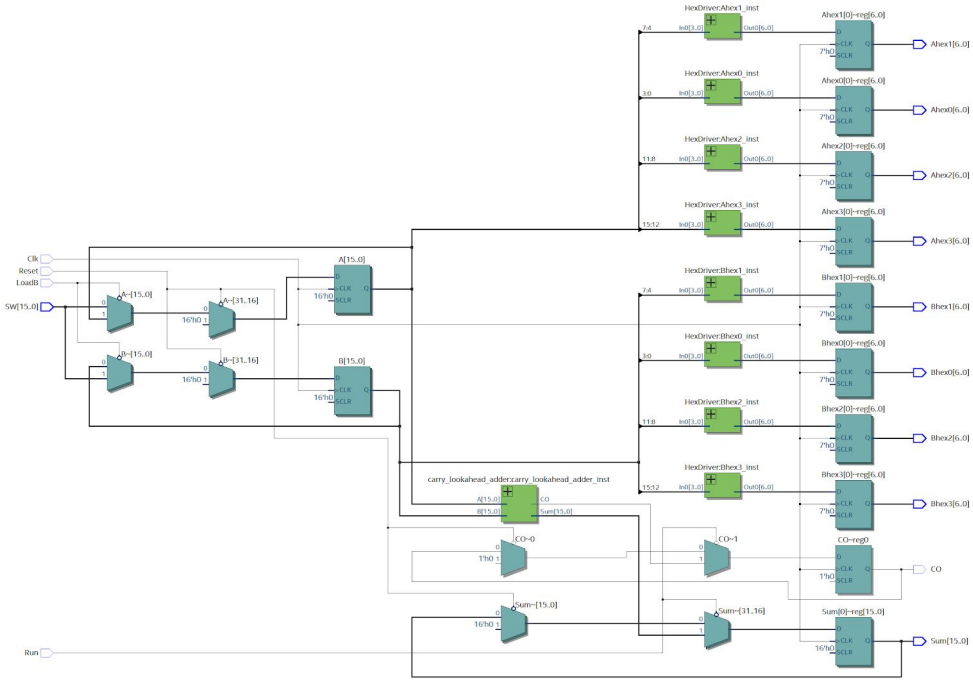
   b. Carry Lookahead Adder

i. The carry look ahead adder taken in a P and G bit string from the full adder with P being A XOR B and G being A AND B. It then generates a Cin signal for each full adder. The adder is connected to 16 full adders.

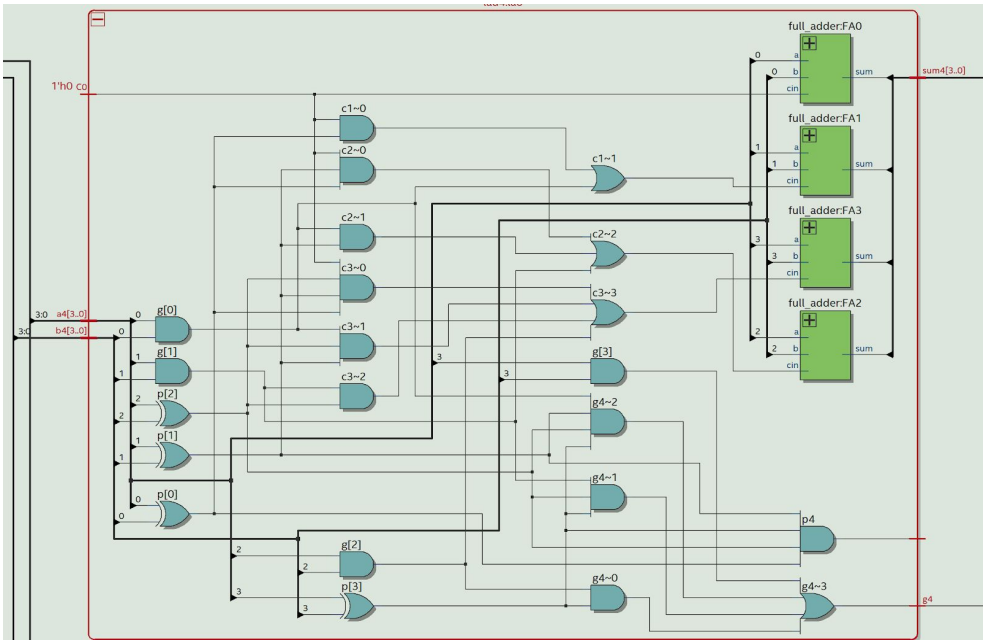
ii. With the P and G signals, we are able to determine ahead of time what the carry bit is for each individual full adder. Combined together with C0, the initial carry in bit, we use complicated logic to output a 1 or 0 for each individual full adder. We know that if there if the G bit of the previous adder is a 1 then the next adder's carry in bit has to also be a 1.

iii. To build the 4x4 bit hierarchy order, we had to first create the 4 bit adder which was 4 full adders in combination with a carry look ahead part. Then we chained 4 of those module together. In between each module, we connected the P and G outputs of one 4 bit module with the next module's carry in port in combination with several logical operations.

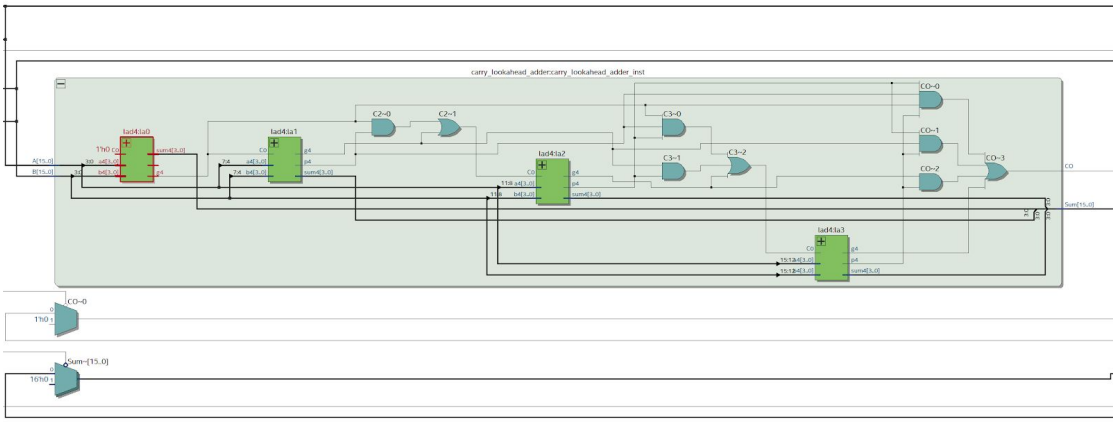
iv. Highest level Block Diagram



v. Lowest level Diagram



vi. Middle level Diagram



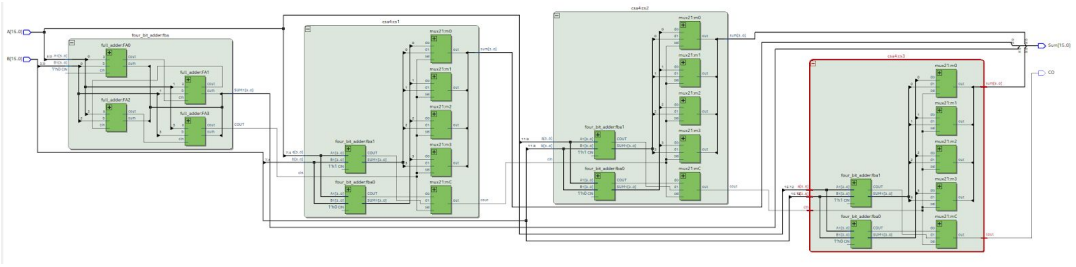
c. Carry Select Adder

i. The 16 bit carry select adder consists of 32 full adders and they are grouped into groups of 8. Each 4 bit carry select adder is split into two groups of 4 full adders, each carrying out a an addition of either Cin being 1 or 0. Then when the Cin signal arrives a MUX is used to choose which answer is used for the final output of the addition. There are a total of 5 MUXes one for each bit of the output as well as one to choose which Cout bit it outputs.

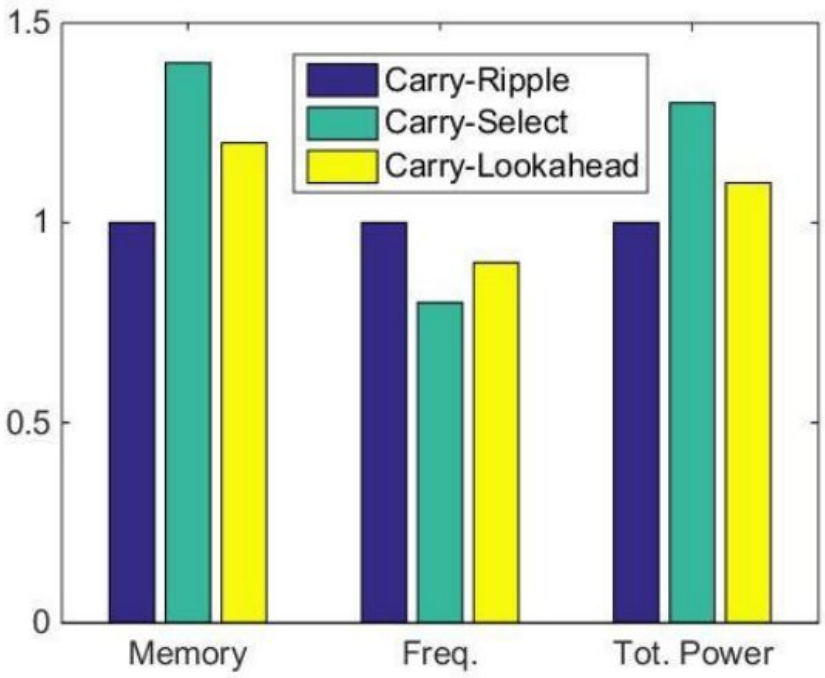
ii. Two separate operations are carried out for the 4 bit version of the carry select adder. Then once the Cin bit comes in, then the correct output for the sum is rapidly outputted instead of waiting for the sum to be computed parallely. Once the Cin bit comes, then 4 bits of the sum comes out parallely. The process is really rapid since we don't have to wait for the answer to be computed, instead,

all the possibilities are calculated and once the first Cout is outputted, the answers are computed 4 bits at a time, much faster than the ripple adder.

iii.



d. The look-ahead adder is the most complex one among those three adders, but it should perform the best among those three adders.



e.

4. Post Lab
   a. The number of LUT combinational components could be more since we didn't use all of the port on each chip that was present and there were a lot more gates in the counters, as well as the MUXes. As for the registers, I think that there were way less registers used in the TTL design since we would have 8 * 2 + 3 registers in total as compared to the 32 flipflops that the SystemVerilog version had. As for memory both would be the same, 0 bits since we did not have amy sort of memory build into this design other than the registers which can hold up to 16 bits combined.

LUT	72
DSP	0
Memory (BRAM)	0
Flip-Flop	43
Frequency	370.64 MHz
Static Power	98.55 mW
Dynamic Power	0.00mW
Total Power	158.63 mW

Post-lab question2: This would not be ideal since there are only 4 bits that are computed parallely. To get the optimal design we would have to graph the different designs. We would have to figure out the amount of time that it takes for a single operation on a full adder as well as the time it takes for a MUX to output the correct output. Then we can compare how many full adders we can have in series with the amount of MUXes to calculate the optimal combination of full adders in a single module and get the optimal design.

Post-lab question3: according to the tables provided below, the maximum operating frequency of carry look-ahead adder is higher than other two's. Also, it consumes more power than the other two's. Also, the number of LUT of look- ahead adder is bigger than the other two adder's.

Ripple adder

LUT	114
DSP	0
Memory (BRAM)	0
Flip-Flop	105
Frequency	189.93 MHz
Static Power	98.49mW
Dynamic Power	0.00mW
Total Power	138.27mW

Carry look_ahead adder

LUT	135
DSP	0
Memory (BRAM)	0
Flip-Flop	105
Frequency	312.5 MHz
Static Power	98.50mW
Dynamic Power	0.00mW
Total Power	144.78mW

Carry select adder

LUT	125
DSP	0
Memory (BRAM)	0
Flip-Flop	105
Frequency	198.89 MHz
Static Power	98.50mW
Dynamic Power	0.00mW
Total Power	139.77mW

5. Conclusion

   The biggest bug we meet during the lab is that the mux function we wrote was not correct at the first time and this caused some trouble during the debug part since we thought that our design was correct, and it took us longer than expected to finish the process. Overall, the lab was well constructed and it was easy to understand once we under the syntax of SystemVerilog as well as how the code gets translated into hardware. I think that more practice in SystemVerilog would prefered, but overall it was a good lab.