Experiment 4
Introduction to System Verilog, EDA, and 16 Bit Adders
ECE 385

Jordan Tan JTAN41, Jeffrey Huang JHUANG61

February 15, 2017

Date Performed: February 15, 2017

1 Objective

The objective of this experiment is to introduce system verilog by coding the serial logic processor from lab 3 in verilog as well as creating three different adders in verilog. The adders are the carry-select adder, carry-lookahead adder, and the ripple adder.

2 8-bit Processor Block Diagram

The 8 bit processor block diagram

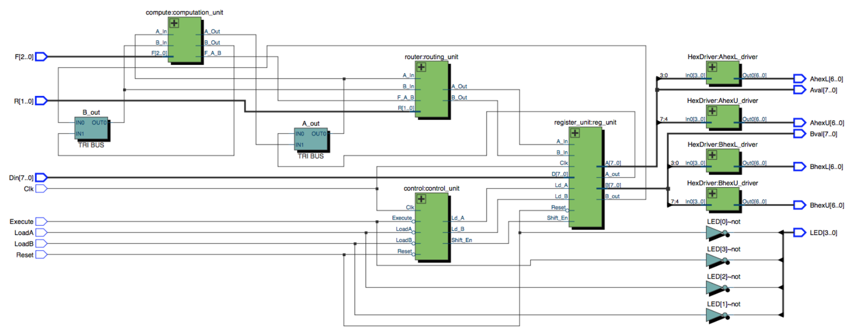
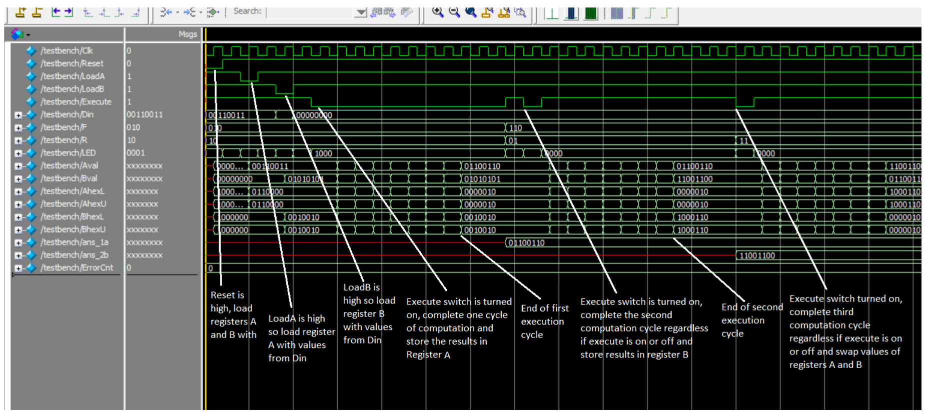


Figure 1: 8 bit block diagram.

3 8-bit Annotated Processor

This is the 8-bit Annotated Processor Simulation



Reset is high, load registers A and B with

LoadA is high so load register A with values from Din

LoadB is high so load register B with values from Din

Execute switch is turned on, complete one cycle of computation and store the results in Register A

End of first execution cycle

Execute switch is turned on, complete the second computation cycle regardless if execute is on or off and store results in register B

End of second execution cycle

Execute switch turned on, complete third computation cycle regardless if execute is on or off and swap values of registers A and B

Figure 2: Annotated Processor Simulation.

4 Changing from 4 bit to 8 bit

In order to change the provided code from a 4 bit logic processor to an 8 bit we have to change the data sizes in the register unit. We also had to change the size of the input size from the switches. We then had to change the size of the output value to the LED in the processor unit. Lastly we had to expand the number of states in the control logic. In order to operate on the additional 4 bits, we had to add 4 more states.

4.1 Prelab Design Considerations

Carry-Ripple	Memory 0	Frequency 154.3 baseline ,	Total Power 148.6
Carry-Select	Memory 0	Frequency 242.3(1.57),	Total Power 148.6
Carry-Lookahead	Memory 0	Frequency 223.4(1.45),	Total Power 148.6

5 The Adders

The adders at the base level take in two values which are the numbers that you are trying to add. It also takes in another bit which is the Carry-in bit. The adder adds this carry-in bit to the resulting value of the addition of the two input values. The adder has two bits, the output and the carry-out bit. Using this you can create a larger adder but putting multiple adders together.

5.1 Carry Ripple Adder

We created this circuit by first building a 4 bit adder after connecting 4 single bit adders together. We connected the carry out bit from one into another.

After creating this 4 bit adder, we connect 4 of these modules together into a larger 16 bit adder using the same method we used to create the four bit adder.

There are problems with this adder because of the propagation delay since every adder relies on the output of the previous adder in serial. We can get around this by using another layout called the Carry Lookahead Adder

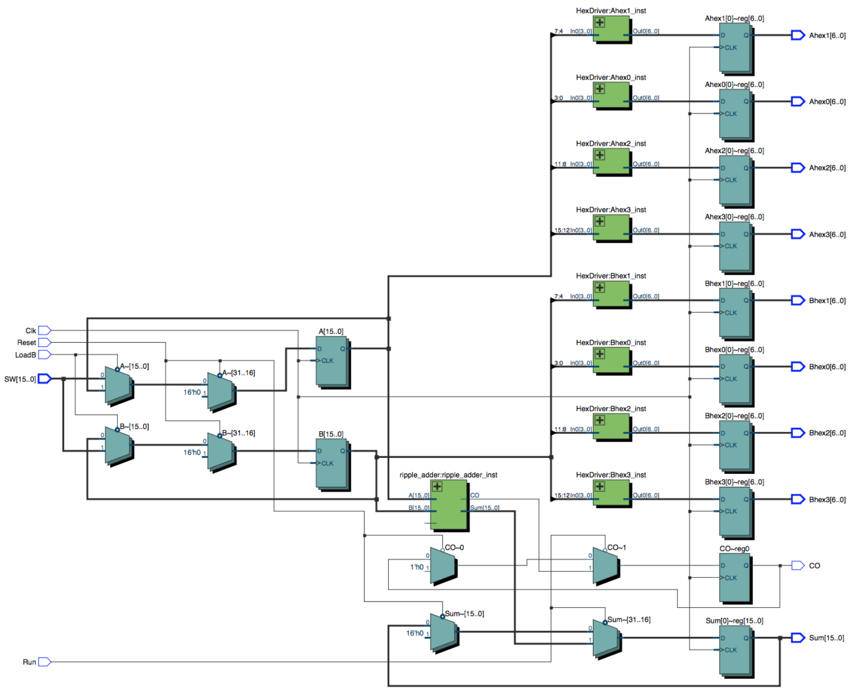


Figure 3: Carry Ripple Adder.

5.2 Carry Lookahead Adder

We created this circuit in a similar to the carry ripple adder. We first creating a smaller 4 bit adder and then extended it by connecting 4 in a line. Instead of hooking things in serial, it uses the idea of propagation and generation of a carry bit. The propagation and generation bits are fed from every adder into a control panel the performs the following expressions

$$G(A,B) = A \cdot B \tag{1}$$
$$P(A,B) = A \oplus B \tag{2}$$
$$C_{i+1} = G_i + (P_i \cdot C_i) \tag{3}$$

This C_i value is fed into the next adder. The adder can use the preliminary propagation and generation bits to create the next propagation and generation bits. By passing these bits down before the actual value is calculate, the adders can work in a more parallel way. While creating the controller that generates the carry in bit for the next adder will take more logic gates, this scheme will be faster because you can calculate propagation and generation bits over a larger span of bits.

The advantage of this adder is similar to the difference between doing operations in parallel vs serial.

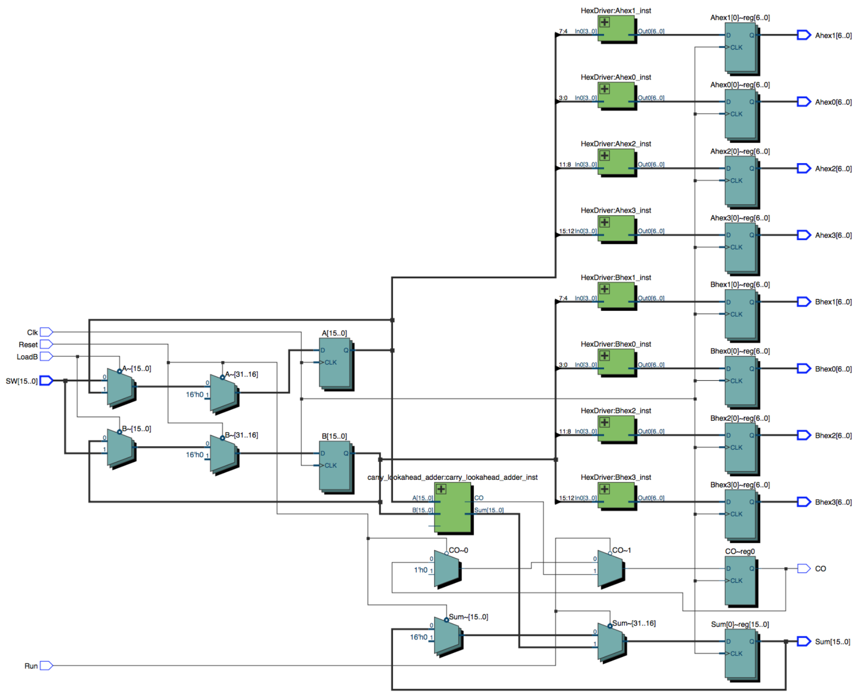


Figure 4: Carry Lookahead Adder.

5.3 Carry Select Adder

The carry select adder uses a different approach. This circuit will pre-calculate two sums, one with and one without the carry in bit. These will feed into the multiplexer and when the real carry in bit arrives then select it as the real sum. We used the same design, 4x4 CSA, to create this circuit.

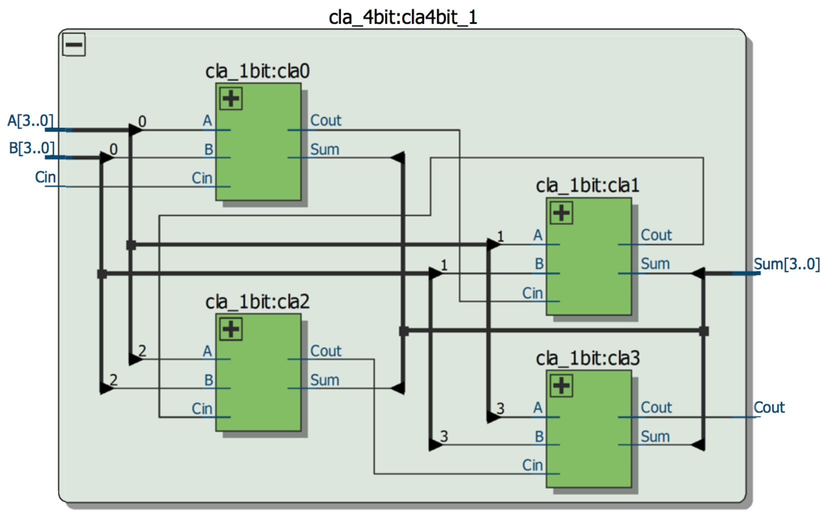


Figure 5: Carry Lookahead Block.

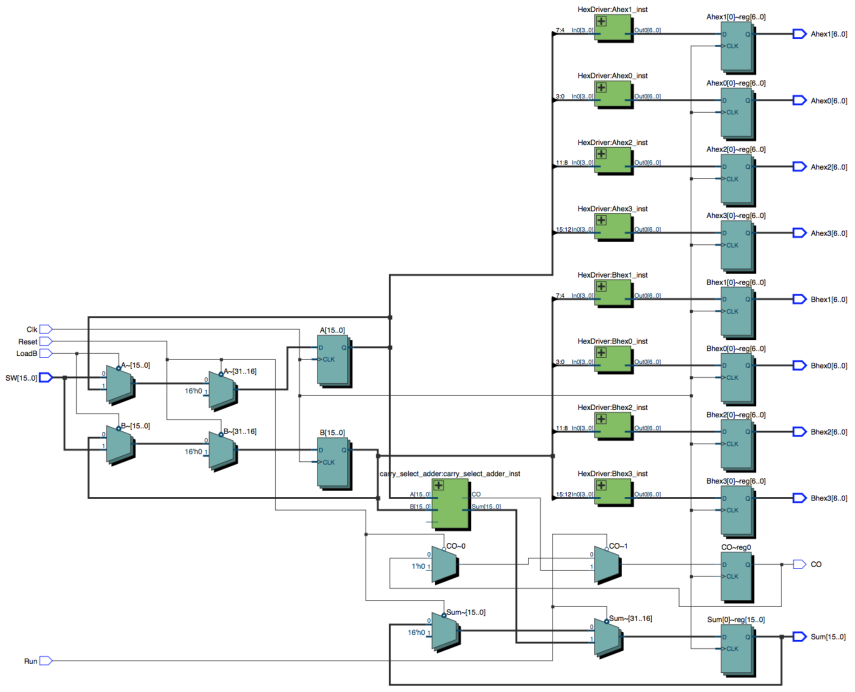


Figure 6: Carry Select Adder.

5.4 Performance tradeoffs between the adders

The main performance differs between the three adders. The ripple adder is slower than the lookahead adder while the lookahead adder is slower than the select adder. This difference boils down to the way that the adders are propagating the carry out bit. Since the lookahead and select adder pre computes the carry out bit, they are able to perform the addition faster. Ranked by complexity, the ripple adder is the simplest, followed by the lookahead adder, and lastly the select adder. This is judged by how many components each adder has.

5.5 Appendix

Module:	cla1bit.sv
Inputs:	Cin, A, B
Outputs:	Sum, Cout
Description	This is the CLA adder for 1 bit
Purpose:	The most basic version of the carry lookahead adder, this is used together with other adders to create a bigger adder.

Module:	cla4bit.sv
Inputs:	Cin, [3:0]A, [3:0]B
Outputs:	[3:0]Sum, Cout
Description	This is the second stage of the carry lookahead adder and works on 4 bits
Purpose:	This is the second biggest building block of the adder and is used together to create a 16-bit adder.

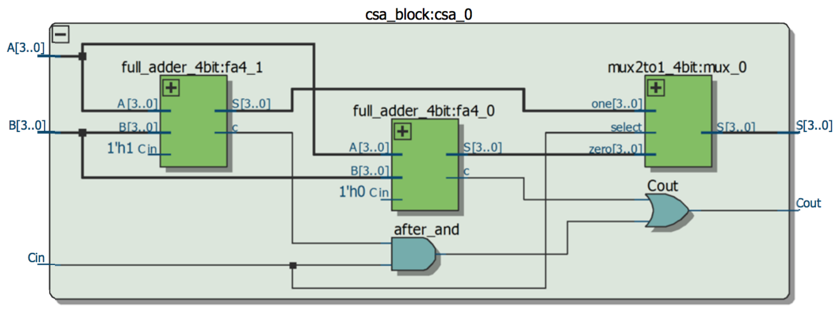


Figure 7: Carry Select Adder Block.

Module:	carrylookaheadadder.sv
Inputs:	Cin, [15:0]A, [15:0]B
Outputs:	[15:0]Sum, Cout
Description	This is the final block of the carry lookahead adder and works on 16-bits
Purpose:	Adds two numbers.

Module:	csablock.sv
Inputs:	Cin, [3:0]A, [3:0]B
Outputs:	[3:0]Sum, Cout
Description	A block of the carry select adder and adds 4 bit values
Purpose:	A building block of the full adder.

Module:	carryselectadder.sv
Inputs:	Cin, [15:0]A, [15:0]B
Outputs:	[15:0]Sum, Cout
Description	A full adder that uses the carry select adder algorithm
Purpose:	Adds two numbers.

Module:	rippleadder.sv
Inputs:	Cin, [15:0]A, [15:0]B
Outputs:	[15:0]Sum, CO
Description	The full ripple adder
Purpose:	Adds two numbers.

Module:	fulladder.sv
Inputs:	x, y, z
Outputs:	s, c
Description	This sets the s and c bits for the full adder
Purpose:	Used in the full adder block.

The IQT Design is better than the TTL design since it is easier to debug and test the IQT design. The chips, wiring, and breadboard are all points of failure in the TTL design and even if the design is correct, the circuit could not work because of any one of these components. In addition to that, testing requires physically turning switches, while the IQT design can be tested automatically. The compile time is a small, but notable, drawback of the IQT design

5.6 Post Lab Questions

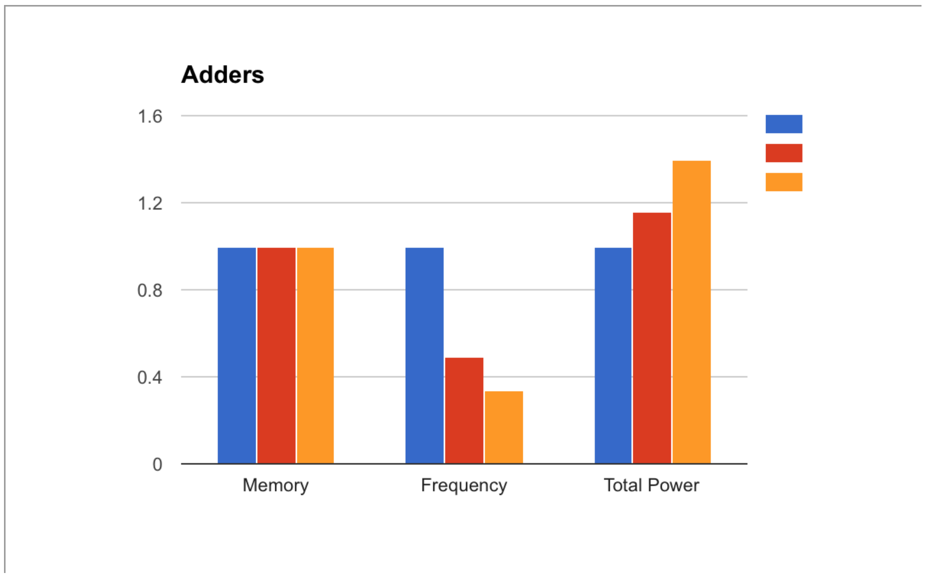


Figure 8: Blue - Carry Ripple adder, Red - Carry Select adder, Yellow - Carry Lookahead adder.

6 Fitter report

.	Carry-Ripple	Carry-Lookahead	Carry-Select
LUT	114	130	160
DSP	0	0	0
Memory	0	0	0
Flip Flop	105	105	105
Frequency	62.32	65.71	93.45
Static Power	98.51	98.54	98.52
Dynamic Power	2.97	2.76	5.51
Total Power	153.81	153.75	155.54

The resource breakdown is comparable to the theoretical design expectations. The lookahead adder and select adder has a higher operating frequency compared to the ripple adder due to optimizations. The Carry Select adder consumes the most power and is the fastest as expected. This is expected because it has more componenets in the design.

7 Bugs

We didn't have any major bugs during this lab.

8 Conclusion

Overall this lab was straightforward. The instructions are clear and there weren't that many bugs that we ran into.