EXPERIMENT # 4
Introduction to SystemVerilog, FPGA, CAD, and 16 bit Adders

Name: Chuangy Zhang
NetID: Czhan30
Name: Homero Vazquez
NetID: hvazqu6

Introduction:

In this experiment we were able to learn and utilize the Quartus and ModelSim Tools and we also learned how to write SystemVerilog to create circuits and map them on an FPGA. We utilize the tools from before to extend a 4-bit processor into an 8-bit processor. We also learned about 3 different types of adder and created them using System Verilog. The adders that we created were Ripple Carry Adder, Carry Select Adder and Carry Lookahead Adder. We were able to compare the performance between each adder.

Serical Logic Processor:

The Serial Logic Processor does logical operations between registers. Some of the operations that it supports are OR, AND, NOR, NAND, XOR etc. and then it routes back the results to the desired registers.

Ripple Carry Adder:

This adder cascades full adders and does the computations bit by bit. The critical path of this adder starts at Cin and ends at Cout. The Cout of the full adders will be the Cin of the following full adders until the adder ends.

Carry Lookahead Adder:

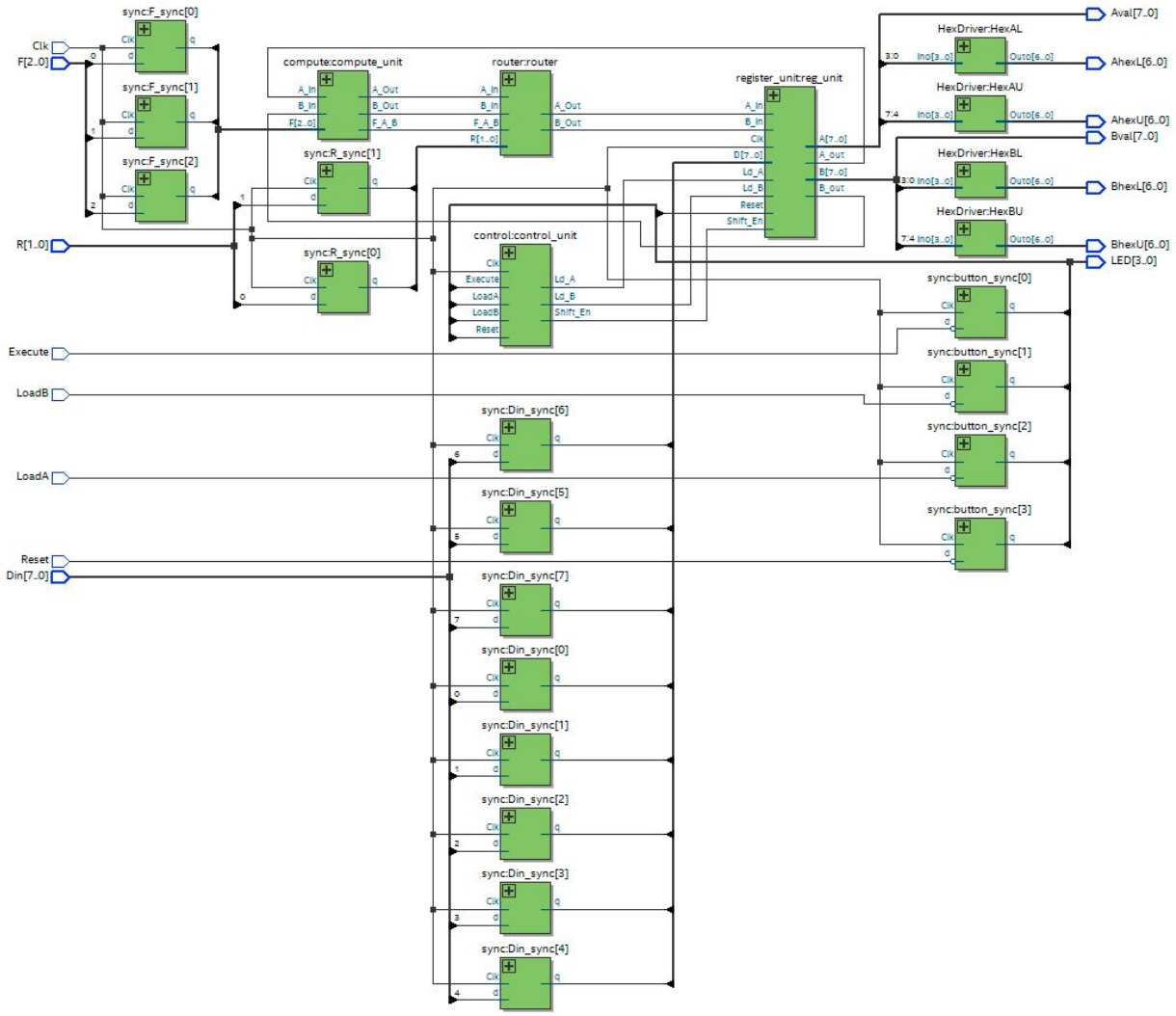
This adder is made up of full adders containing Propagate and Generate signals which allows the adder to calculate each bit independently from previous bits. This adder has less delay than the Ripple Carry Adder

Carry Select Adder:

This adder is built using multiplexers and full adders to pre-compute results for Cin = 0 and Cin = 1 therefore reducing the propagation delay.

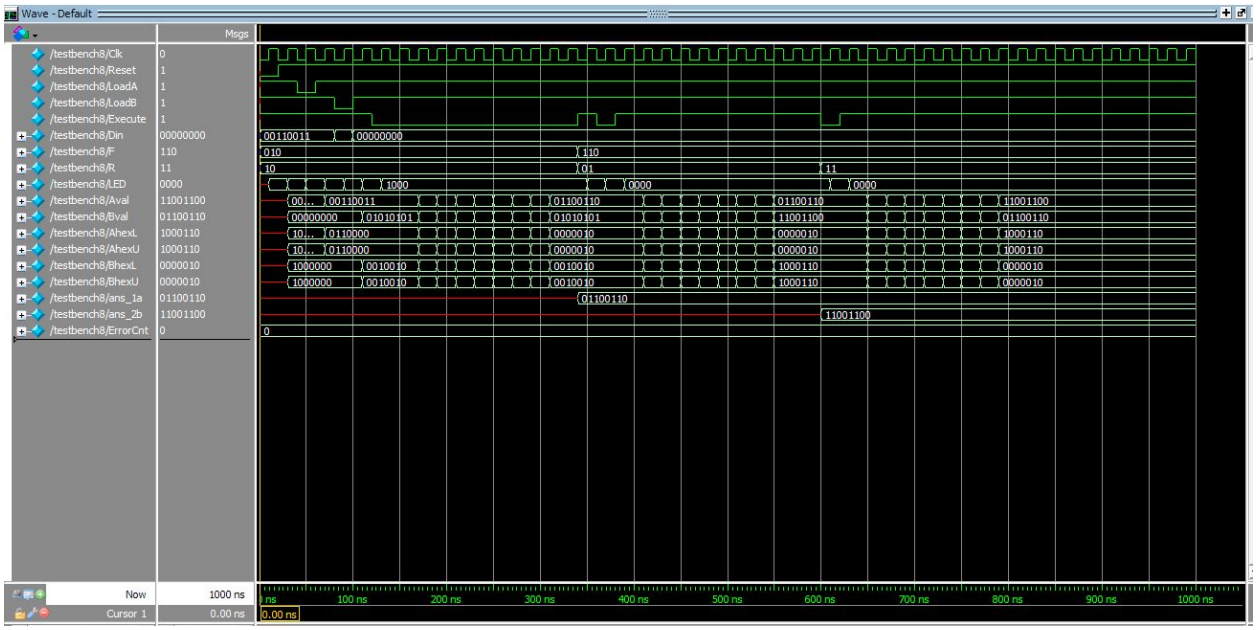
Part 1 - Serial Logic Processor:

a. Block Diagram



b. The things that we did to convert the 4-bit Processor into an 8-bit Processor was changing the size of the register from 4 to 8 bits. We also did modifications to the number of states in the FSM.

c. Simulation



Part 2 - Adders:

a. Ripple Carry Adder

i. If we look into the N-bits Ripple Carry Adder we have N full adder, and each adder consists of 3 inputs A,B and Cin. It also consists of 2 outputs Sum and Cout. Each full adder has its Cin connected to the Cout of the previous one. The output Sum of each full adder will represent the bit of the computation.

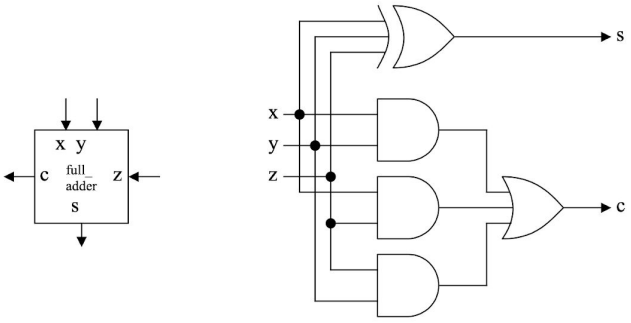


Figure 2: Full-Adder Block Diagram

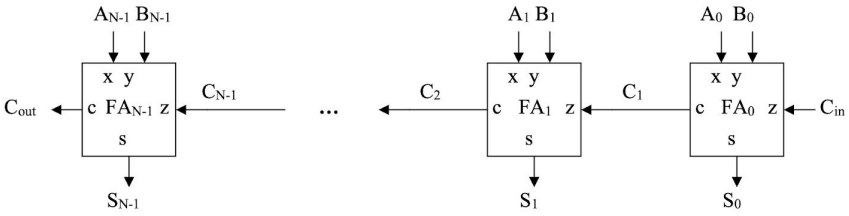
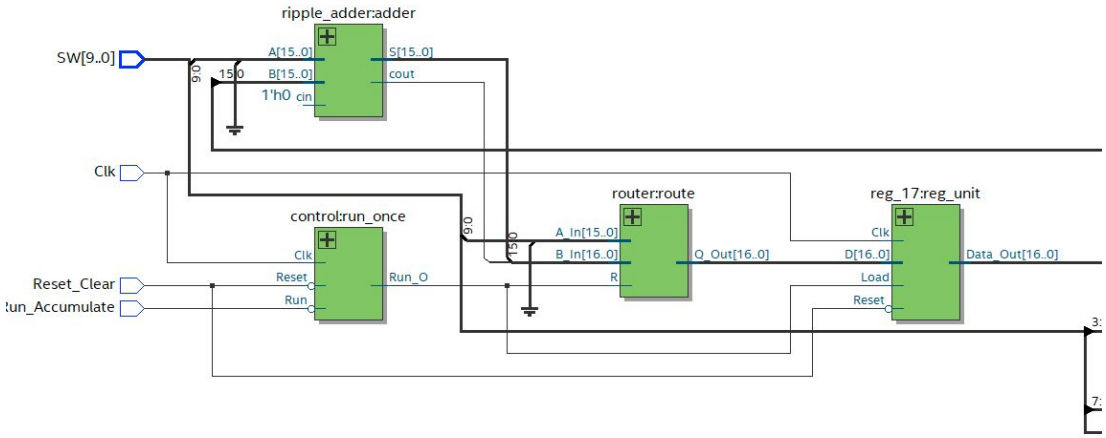


Figure 3: *N*-bit Carry-Ripple Adder Block Diagram

ii.



b. Carry Lookahead Adder

i. This adder contains Generating and Propagating signals which add more logic to the design. In this architecture the Generating and Propagating signals will be used to predict the carry-out values using immediate inputs of A and B. This adder contains 3 inputs A,B and Cin and 3 outputs P, G and Sum.

ii. G is 1 if and only if A and B are 1 which is analogous to the Cin signal. The logic expression for G would be G = AB. P is 1 when either A or B is 1. Therefore we use an

XOR for that expression: P = A XOR B, then Ci+1 = Gi + (Pi Ci) will compute the carry out for the next full adder.

iii. What we did to implement this design was to divide 16 bits into groups of 4 bits. From that we put each group into a 4-bit CLA. Then we cascaded the four 4-bit CLAs by connecting the Cout from the previous one to the Cin of the next one.

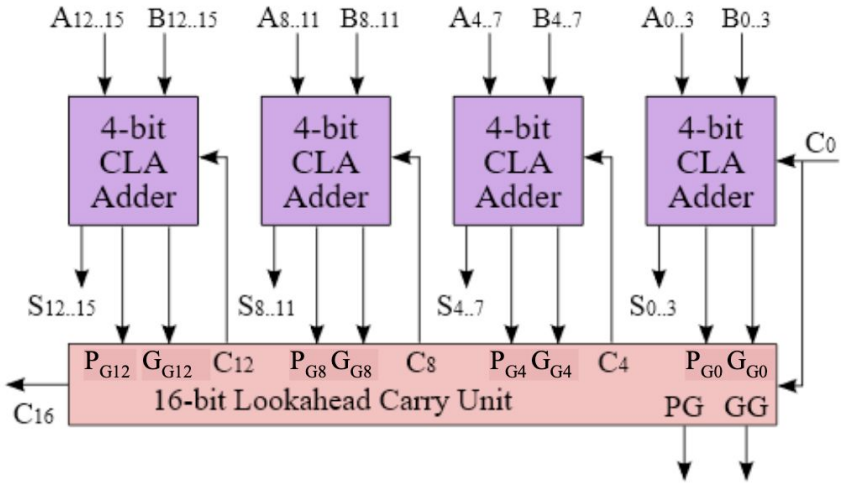
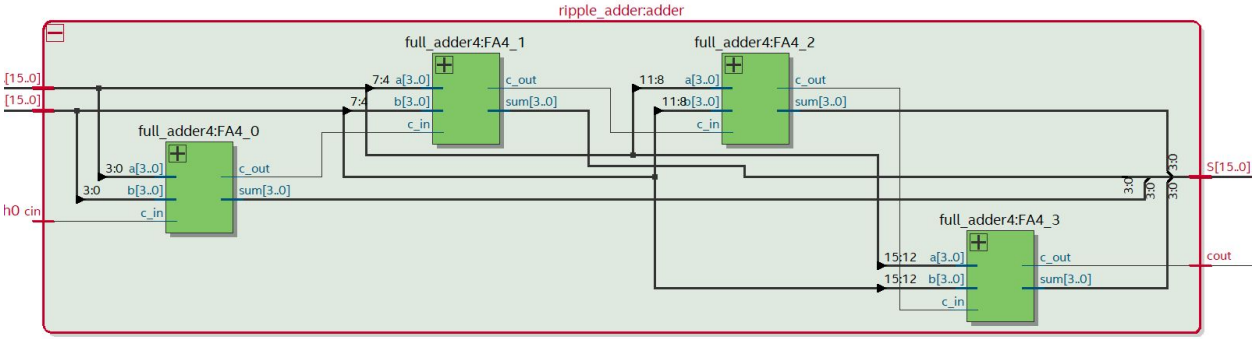
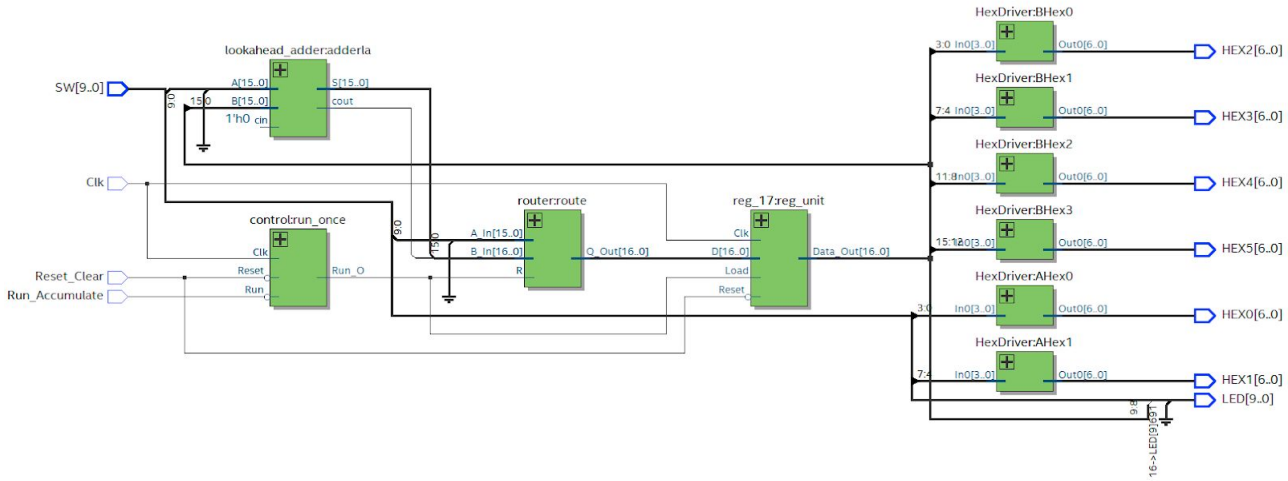


Figure 5: A 4x4-bit Hierarchical Carry-Lookahead Adder Block Diagram





c. Carry Select Adder

i. This adder has doubled the adders as the previous ones and it also includes a multiplexor to select depending on Cin. The CSA pre-computes the result for Cin either equal to 1 or 0 and then once the inputs are ready it will select the right result with the MUX.

ii. The CSA consists of 2 full adders and a MUX. Each adder computes the sum and the Cout with an assumption of Cin either 1 or 0 one for each adder. Once the real values arrive the corresponding Sum and Cout are selected.

.

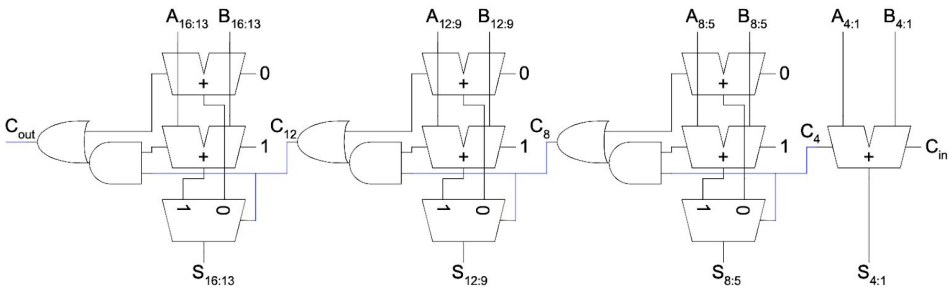
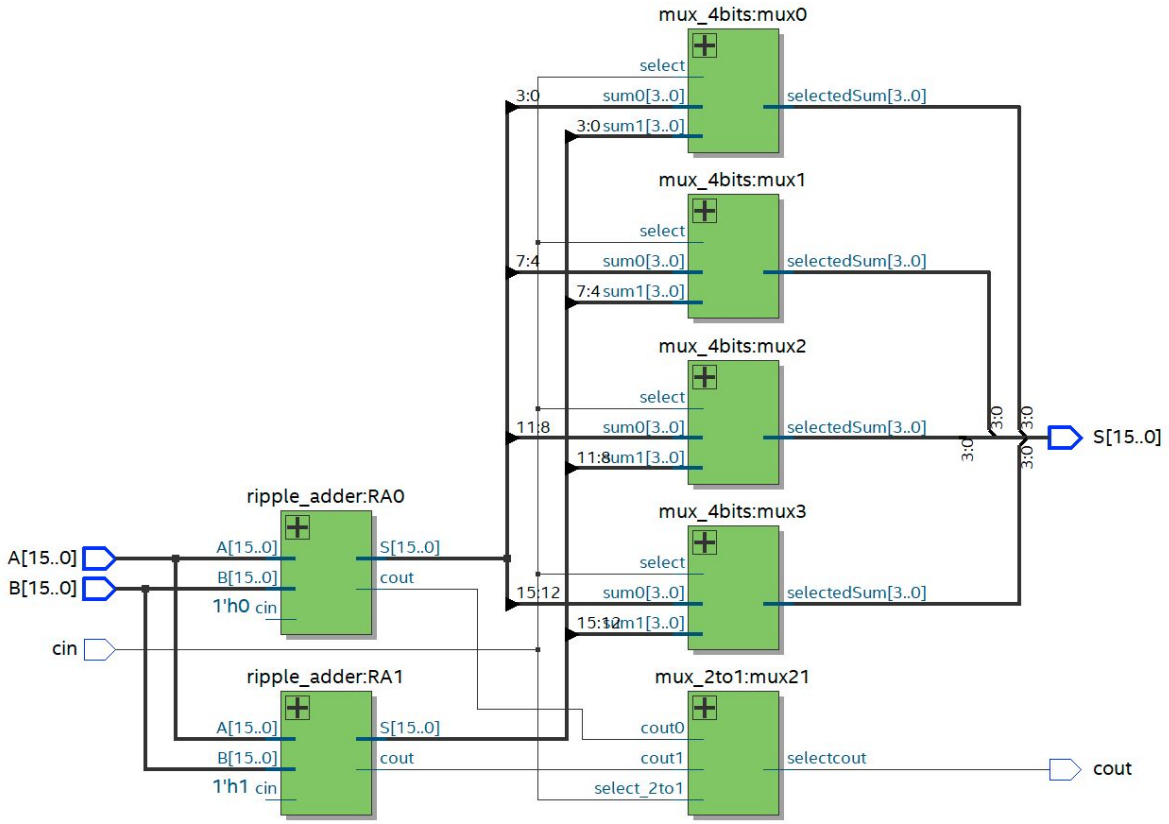


Figure 5: 16-bit Carry-Select Adder Block Diagram



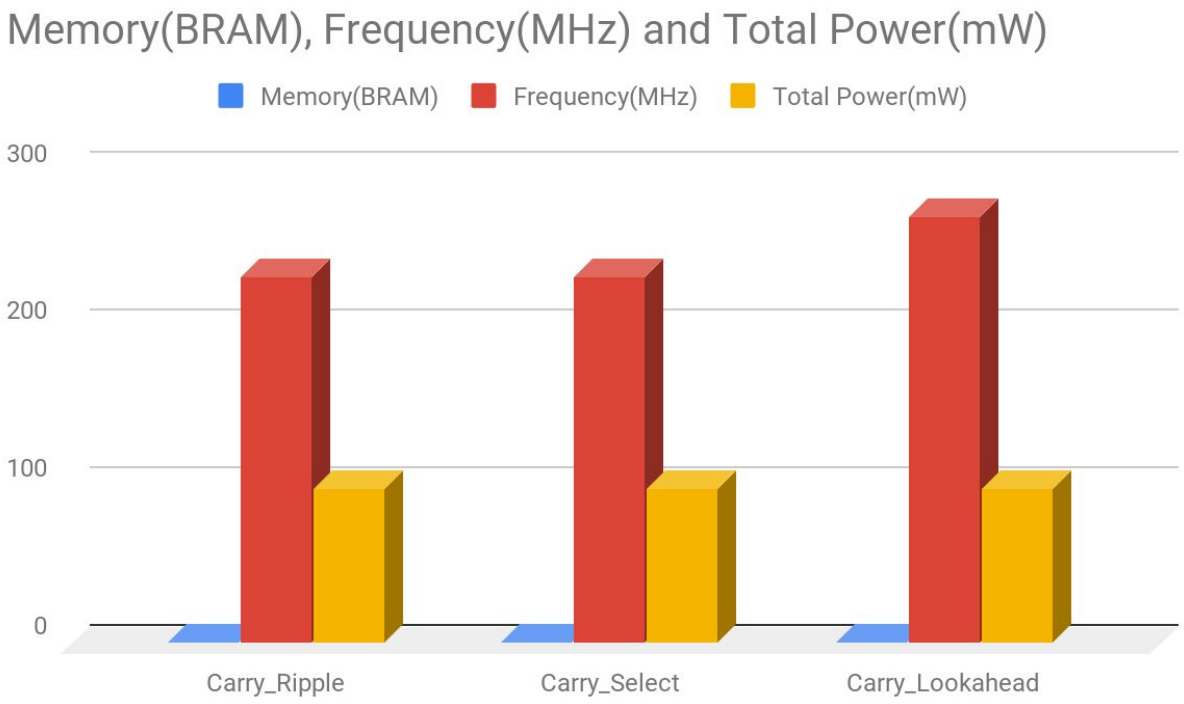
d. Describe at a high level the area, complexity, and performance tradeoffs between the adders.

Type of Adder	Area	Complexity	Performance
Ripple-Adder	small	simple	slow
LookAhead Adder	Medium	complex	fast
Carry Select Adder	large	medium	slow

The area of ripple adder is very minimal in comparison with other two adders and its complexity is simple but it has a very large latency and slow if we are doing a lot bit computation. The lookahead adder has the small area in comparison to the Carry_select adder but lookahead adder has the fast performance and we don't need to wait the carry bit at the most significant bit. Its complexity is more difficult than the other two adders. Carry select adder is the larger area and also has the slow performance speed.

e. Document the performance of each adder by creating a graph as specified in Prelab part C (page 4.6).

	Memory(BRAM)	Frequency(MHz)	Total Power(mW)
Carry_Ripple	0	232.02	98.71
Carry_Select	0	232.02	98.71
Carry_Lookahead	0	270.64	98.71



Post-lab Questions:

1. From what we see comparing this lab and lab 3 we believe that the design in lab 3 was better because of the mealy state machine. We only had two cycles and compared with what we have in the design for lab 4 we have a lot more cycles. Because in lab 4 there are more cycles we will need to use more flip flops usage. But because of the Mealy state machine in lab two then we need to use more logic which means more LUT usage. Therefore there is not a better design but it depends on the tradeoff between flip flops and LUT usage and what fits better for what you will use it.

8 bit processor usage

LUT	73
DSP	0

MEMORY	0
FLIP_FLOP	43
Frequency	77.83 MHz
Static Power	89.97 mV
Dynamic Power	2.03 mV
Total Power	105.51 mV

2. The 4x4 hierarchy CSA might not be ideal. The ideal grouping would be one where each group has the same delay as the previous group plus the MUX delays in that group which are on the critical path.

3. Design Statistics

Selecter adder

LUT	69
DSP	0
MEMORY	0
FLIP_FLOP	20
Frequency	232 MHZ
Static Power	89.94 mV
Dynamic Power	8.76mV
Total Power	98.71 mV

Lookahead Adder

LUT	91
DSP	0
MEMORY	0
FLIP_FLOP	20

Frequency	270.64mHz
Static Power	89.94mv
Dynamic Power	8.67mv
Total Power	98.61mv

Ripple-adder

LUT	79
DSP	0
MEMORY	0
FLIP_FLOP	20
Frequency	232.02mHz
Static Power	89.94mv
Dynamic Power	8.76mv
Total Power	98.71mv

The resource breakdown comparison from the plot makes sense. The maximum operation frequency of the carry-lookahead adder is higher than the carry-ripple adder because we are parallel computing the carry bits and allowing it to shorten its critical path. . These three designs are consuming a silmarly amount of power, which surprised. We thought the select adder consumes more power than those other two designs. We think the reason behind the power consumption is not often can be determined by the small different amount of logic therefore it is not a significant difference in power consumption.

Conclusion:

a. One of the main bugs that we encountered was in the FSM. We were able to multiply negative values but when we multiplied positive values we would get a wrong result. What we discover is that in our FSM we had an addition before the last shift when M = 0. After changing that logic we were able to get the right results with any type of values.
b. It was somehow hard to do the simulation so maybe more help with that would be nice. For example we didn't know that we can run each state separately also we didn't know that we can print things in the terminal

c. Overall it was a really involved and interesting experiment . Now I understand why an ALU doesn't perform multiplication. It is really complex to do it in 1 cycle.