ECE 385

Fall 2022

Experiment #3

Lab 3

Introduction to system Verilog, FPGA, CAD, and 16-bit Adders

Haocheng Yang((hy38)
Yicheng Zhou(yz69)

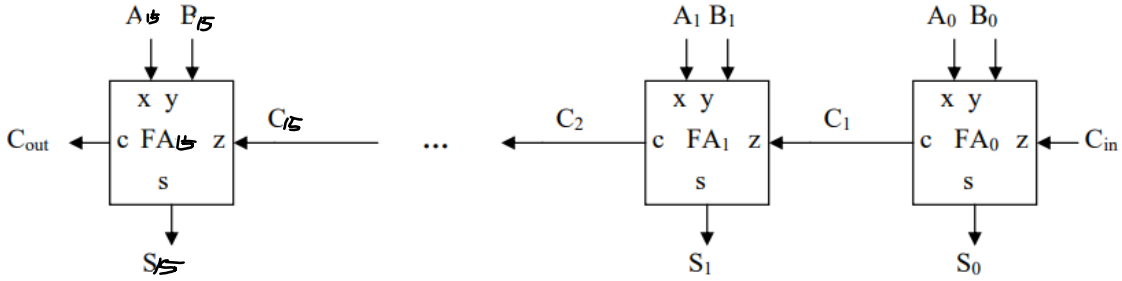
Introduction

In this lab we implemented different 16-bit adders with different design but performs the same function where it takes two 10-bit input, (which are extended to 16-bit in the register) and output the 16-bit sum.

Adders:

Ripple Carry Adder:

1. A 16-bit ripple carry adder consist of 16 1-bit carry adder (1-bit output and 1 carry on). The 1-bit carry adder takes 1 bit from A and B and a carry-on bit from previous adder in sequence.
2. Block diagram:



Carry Lookahead Adder:

1. Instead of waiting for the carry-in values, the carry lookahead adder tries to "predict" the carry in value base on the "G" and "P" logics from the carry-lookahead unit.
2. "G" stands for "generating" meaning that a carry out is generated when both inputs are 1, regardless of carry in, "P" stands for "propagating", meaning that when either A or B are 1, the carry-out has the possibility of being propagated. While trying to eliminate lag from rippling, we can determine the carry out for each bit without waiting for carry in through deducting and expanding the expression for each "P" and "G" to only consists of the previous "P" and "G" as shown below:

$$C_0 = C_{in}$$
$$C_1 = C_{in} \cdot P_0 + G_0$$
$$C_2 = C_{in} \cdot P_0 \cdot P_1 + G_0 \cdot P_1 + G_1$$
$$C_3 = C_{in} \cdot P_0 \cdot P_1 \cdot P_2 + G_0 \cdot P_1 \cdot P_2 + G_1 \cdot P_2 + G_2$$
...

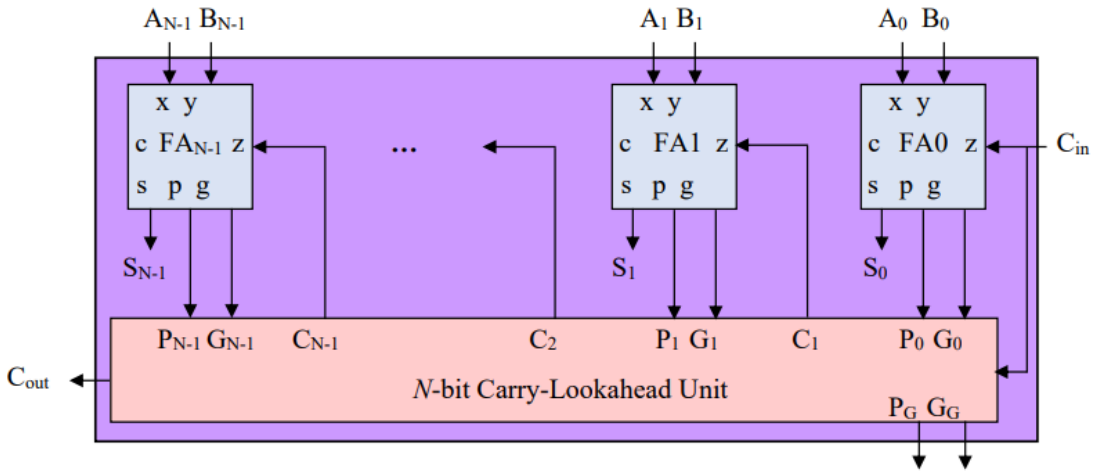


Figure 4: *N*-bit Carry-Lookahead Adder Block Diagram

3. Although Carry-lookahead adder is faster than Carry-ripple adder, the gates required for an explicit 16-bit lookahead adder is too many to be practical, thus we construct 4 4-bit CLA and a hierarchical fashion.
   The fundamental logic of the 4x4 implementation is to treat each 4-bit CLA as a single digit for the hierarchy 4bit CLA.
   For the base adder, we can use the design showcased above to implement "normal" 4-bit CLA, however, to fit our hierarchical design, we need to define the new P_{group} and G_{group} expression to correctly compute 4 digits:

$$P_G = P_0 \cdot P_1 \cdot P_2 \cdot P_3$$
$$G_G = G_3 + G_2 \cdot P_3 + G_1 \cdot P_3 \cdot P_2 + G_0 \cdot P_3 \cdot P_2 \cdot P_1$$

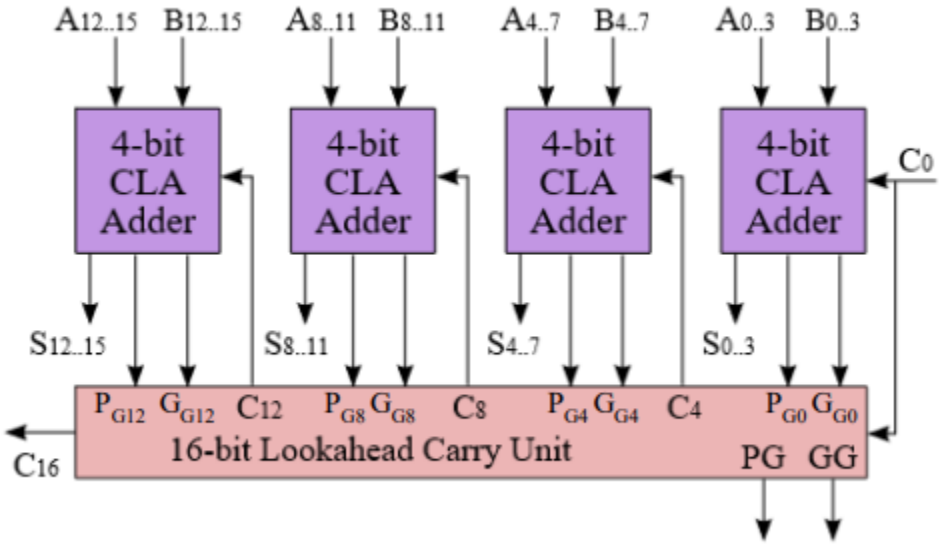
Then, we generate C_{ins} from the 4-bit CLA using P_{Gs} and G_{Gs} with the formula shown below:

$$C_4 = G_{G0} + C_0 \cdot P_{G0}$$

$$C_8 = G_{G4} + G_{G0} \cdot P_{G4} + C_0 \cdot P_{G0} \cdot P_{G4}$$

$$C_{12} = G_{G8} + G_{G4} \cdot P_{G8} + G_{G0} \cdot P_{G8} \cdot P_{G4} + C_0 \cdot P_{G8} \cdot P_{G4} \cdot P_{G0}$$

Then we can take a carry-lookahead unit with altered inputs (Pg and Gg) to complete our design as showed below:

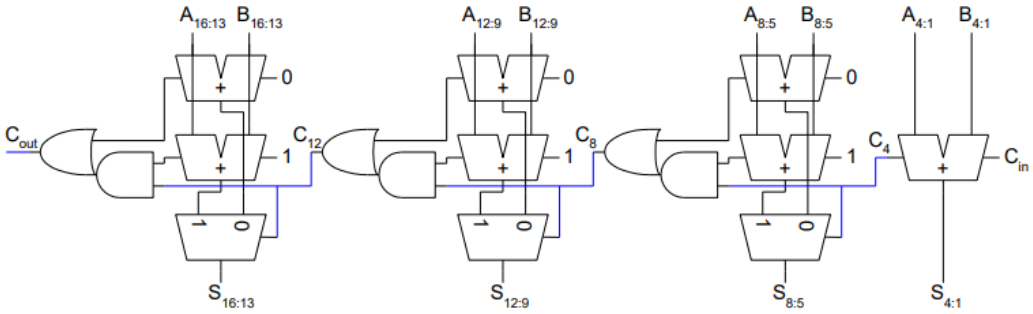


Carry Select Adder

1. The 16-bit carry select adder consists of 32 1-bit adder that are grouped and connected into 2 parallel lines. Each line is similar to a ripple carry adder. Instead of trying to "guess" the carry in, carry select adder (CSA) would compute both scenario and "choose" the output through multiplexer through inputs.
2. CSA is faster due to the ability to do parallel computations. Assuming a 4x4 CSA, we can compute each 4-bit adding simultaneously (both C=0 and C=1) and then base on the C from each 4-bit adder, we go through 5 Mux(s) in parallel to select the correct output.

Since mux gate delay is smaller than 4-bit adder, the whole system is faster.

3. The block diagram is shown below:



.SV Modules

1. Ripple_adder:
   a) Inputs: [15:0]A, [15;0]B, cin,
   b) Outputs:[15:0]S, cout,
   c) This module is independent of clock and S will produce the sum of A and B via ripple adder method.
   d) This module is used to perform 16-bit add calculation

2. lookahead_adder:
   a) Inputs:[15:0]A, [15;0]B, cin,
   b) Outputs:[15:0]S, cout,
   c) This module is independent of clock and S will produce the sum of A and B via carry-lookahead method.
   d) Tis module is used to perform 16-bit add calculation

3. Select_adder:
   a) Inputs:[15:0]A, [15;0]B, cin,
   b) Outputs:[15:0]S, cout,
   c)This module is independent of clock and S will produce the sum of A and B via carry-select method.
   d) This module is used to perform 16-bit add calculation

4. Router
   a) Inputs: R,[15:0]A_In,[16:0]B_In,
   b) Outputs:[16:0] Q_out;

c) This module does not depend on clock and Q_out will produce the result in A or B depending on R.
d) this module is a 17 bit parallel multiplexer.

5. Reg_17:
   a) Inputs: [16:0] Din, Clk,Load,Reset
   b) Outputs:[16:0]Data_Out
   c) This is a positive-edge triggered 17-bit register with asynchronous reset and synchronous load. When Load is hight, data is loaded from Din into the register on the positive edge of Clk.
   d) This module is used to create the registers that store operands A and B in the adder circuit.

6. Hexdriver:
   a) Inputs:[3:0]In0
   b) Outputs:[6:0]Out0
   c) It takes four bits input from in0 and converts to 7 bits for hex display.
   d) This module converts 4bit binary numbers from registers into hexadecimal so correct image is displayed on LED.

7. Control
   a) Inputs: Clk, Reset, Run
   b) Outputs: Run_O
   c) This is a state machine with asynchronous reset, Only in State B (When input Run is High), the output Run_O will be high.
   d) This module is the state machine that gives the execution signal when we press a button and prevents multiple execution if button is not released.

8. Adder4
   a) Inputs:[3:0]A_4,[3:0]B_4 ,c_in_4
   b) Outputs:[3:0] s_4, c_out_4

c) This module is independent of clock and S will produce the sum of A and B
d) This module is used to perform 4-bit add calculation

9. Adder2
   a) Inputs :Clk, Reset_Clear, Run_Accumulate, input[9:0]SW,
   b) Outputs: [9:0]LED,[6:0]HEX0,[6:0]HEX1,[6:0]HEX2, [6:0]HEX3,[6:0]HEX4,[6:0]HEX5,
   c) This is the top level hierarchy that will call on control, router, reg_17,adder, and HexDriver.
   d) This is the top level module that takes in inputs from switches and buttons and send displays to LEDs.
10. CLA_4
   a) Inputs:[3:0] A_4,[3:0]B_4,c_in_4
   b) Outputs:[3:0] s_4, G_out, P_out
   c) This module is independent of clock and S will produce the sum of A and B via lookahed method
   d) This is sub-module of lookahead_adder and is used for computing 4-bit add calculation.
11. CS
   a) Inputs:[3:0]A_4, [3:0]B_4,c_in_4
   b) Outputs:[3:0] s_4
   c) This module is independent of clock and S will produce the sum of A and B via carry select method
   d) This module is a 4-bit carry select adder that is sued in construction of the 16-bit carry select adder.

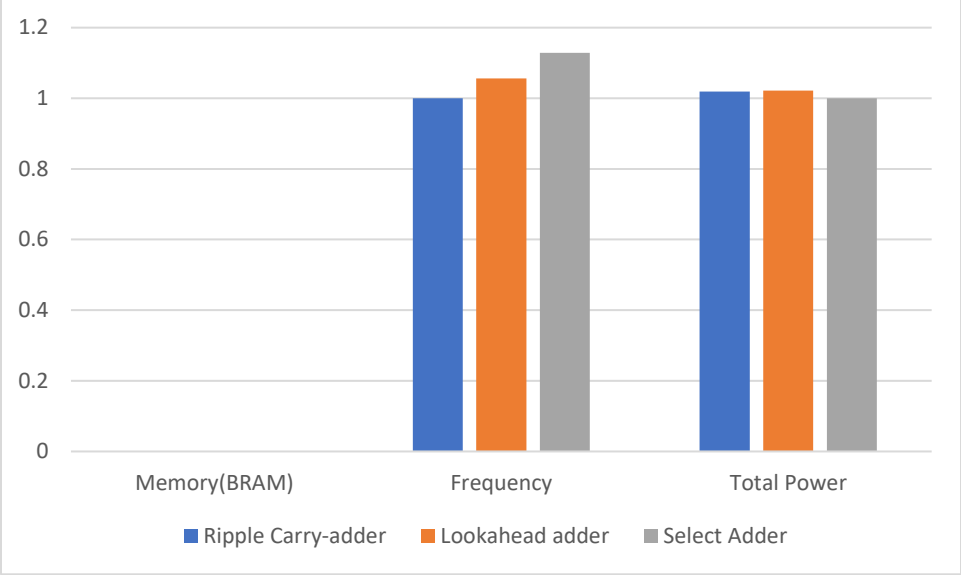
Complexity and performance tradeoffs between adders:

1. The ripple carry-adder: It has the least amount of physical complexity, as a result it is the most compact among the three. However as a trade off, due to its sequential nature, it is the slowest and produces the most amount of lag
2. The carry lookahead adder: It is more complex compared to the ripple carry-adder, thus it takes up more space, however, the computational capacity grows exponentially with time, and is extremely beneficial for large bits operations.

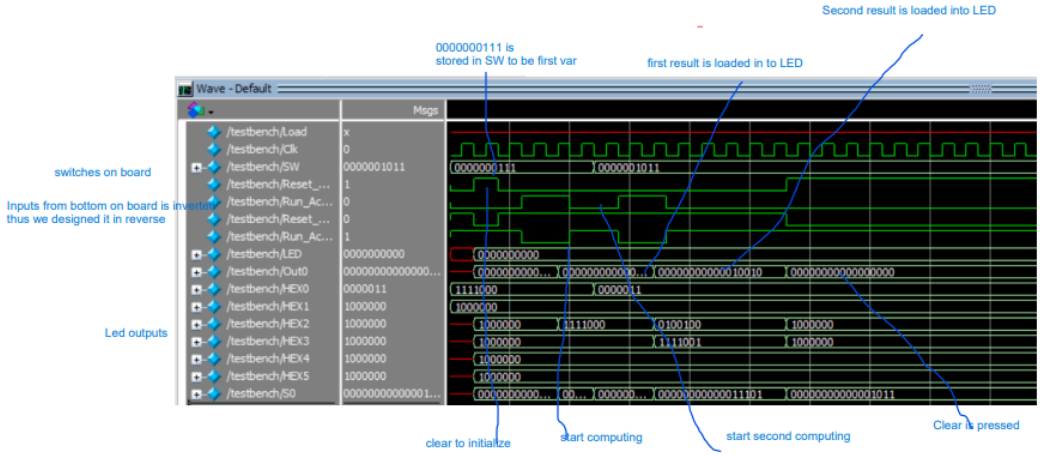
3. The carry select adder. The most complex implementation of the three, and takes up more the twice the amount of space of a ripple carry-adder. However, with the assumption that AxN design is implemented, we can adjust N to modify bits, the capacity and performance would increase linearly.

Documentation of performance:

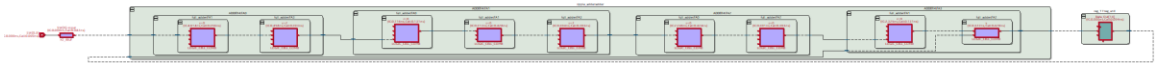
	Ripple Carry-adder	Lookahead adder	Select Adder
Memory(BRAM)	0	0	0
Frequency	1	1.055950479	1.12879393
Total Power	1.018787361	1.021633931	1



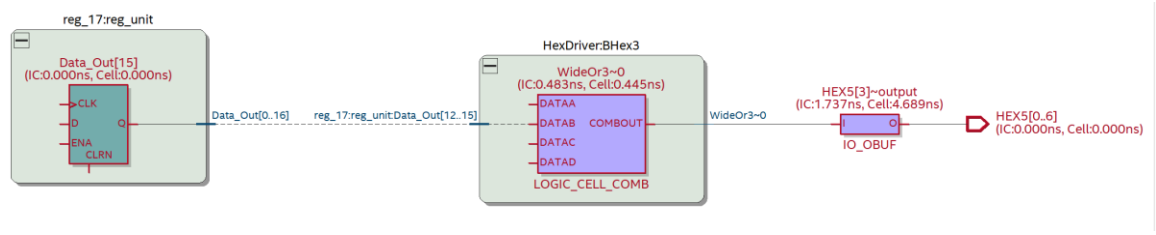
Annotated simulation trace:



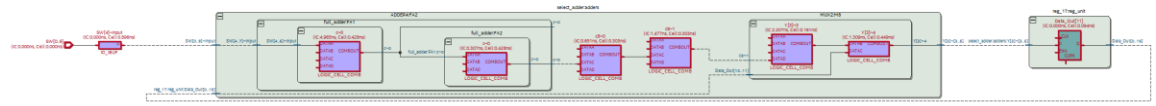
Extra credit:
Ripple adder



Lookahead adder



Select adder



Answers to the post-lab questions:

1. *In the CSA for this lab, we asked you to create a 4x4 hierarchy. Is this ideal? If not, how would you go about designing the ideal hierarchy on the FPGA (what information would you need, what experiments would you do to figure out?)*
   No, in theory we can optimize the CSA even further by implementing different group sizes so that the signal arrive at the mux simultaneously. To do this we need to know the precious gate delays of the logic gates and the mux(s). By starting with smaller groups and gradually increasing group size (since C signal takes time to be computed and travel), we can achieve higher speed. The more bits there are the more pronounce this effect should be.

2. *For the adders, refer to the Design Resources and Statistics in IQT.16-18 and complete the following design statistics table for each adder. This is more comprehensive than the above design analysis and is required for every*

SystemVerilog circuit. Observe the data plot and provide explanation to the data, i.e., does each resource breakdown comparison from the plot makes sense? Are they complying with the theoretical design expectations, e.g., the maximum operating frequency of the carry-lookahead adder is higher than the carry-ripple adder? Which design consumes more power than the other as you expected, why?

The detailed table is shown below:

Ripple Carry-adder	Lookahead adder	Select Adder	
78	86	82	LUT
0	0	0	Memory(BRAM)
20	20	20	Flip-Flop
250.4	264.41	282.65	Frequency(Mhz)
90.15	90.15	90.13	static Power(mW)
11.28	10.92	8.1	Dynamic Power(mW)
143.19	143.56	140.52	Total Power(mW)

Yes the breakdown comparison from the plot make sense. The maximum operation frequency of the carry select adder is the highest and the Ripple carry-adder is the lowest as expected. Also the lookahead adder has the highest power consumption as expected due to its greater number of combinational logic gates. However, even select adder has more LUT, ripple carry-adder consumes more power.

Conclusion:

We did not experience much difficulty when writing the codes itself, however, we did had some trouble trying to test and analyze our design and getting the correct data. This is due to unfamiliarity with quartus prime operation, after many tutorials later we got our desired data.

I think that even in video format the operations of different analyzer in quartus prime is not clearly showcased, otherwise the instructions and explanations from the lab manual is clear.