Performance Analysis of Different Binary Adders

Introduction

In this lab, we constructed the design of three types of 16-bit adders: the ripple carry adder, the lookahead adder, and the carry select adder. The ripple adder is the simplest one of the three; it consists of N full adders, which is the 1-bit version of the binary adder. Each carryout from the full adder goes as a carry-in into the next full adder. The carry lookahead adder uses the carry-in with the inputs of every bit to make predictions about the carry-out. A carry-out is generated when both inputs are 1, or a carry-out could be propagated if either input is 1. The carry select adder uses two full adders and a multiplexer for each bit. One adder computes the carry-out, assuming that the carry-in is one, and the other assumes it's zero. Then, the real carry-in selects the output of the desired full adders in all bits simultaneously. We will discuss each adder in this report regarding area, power, and maximum operating frequencies.

1. Adders:

Ripple Carry Adder

We built the ripple carry adder in a hierarchical manner, meaning that we implemented it using different layers of design. The first layer consists of the full-adder circuit. The second layer is a group of four full adders connected together, meaning that the carry-out (c) from the first full adder would be connected to the carry-in (z) of the second full adder and so on until the fourth full adder. The third layer is four groups of four full adders, as shown in Figure 2, where the carry-out from the first four group of full adders is connected to the carry-in of the second group of full adders and so on, giving a total of 16 full adders, allowing for 16-bit addition. The negative aspect of this design is that each full-adder cell needs to wait for the carry-out of the previous full-adder cell, which costs a lot of time as we want to add larger numbers. On the other hand, the positive aspect is that it's a simple design that uses minimum resources to implement.

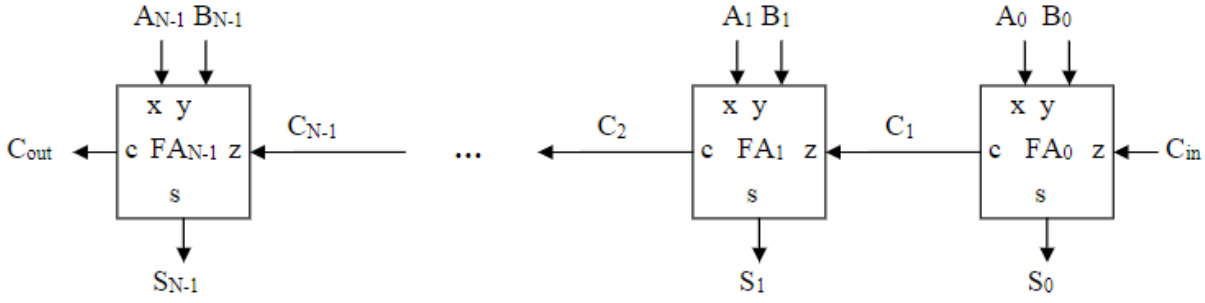

Figure 1: Ripple Carry Adder block diagram

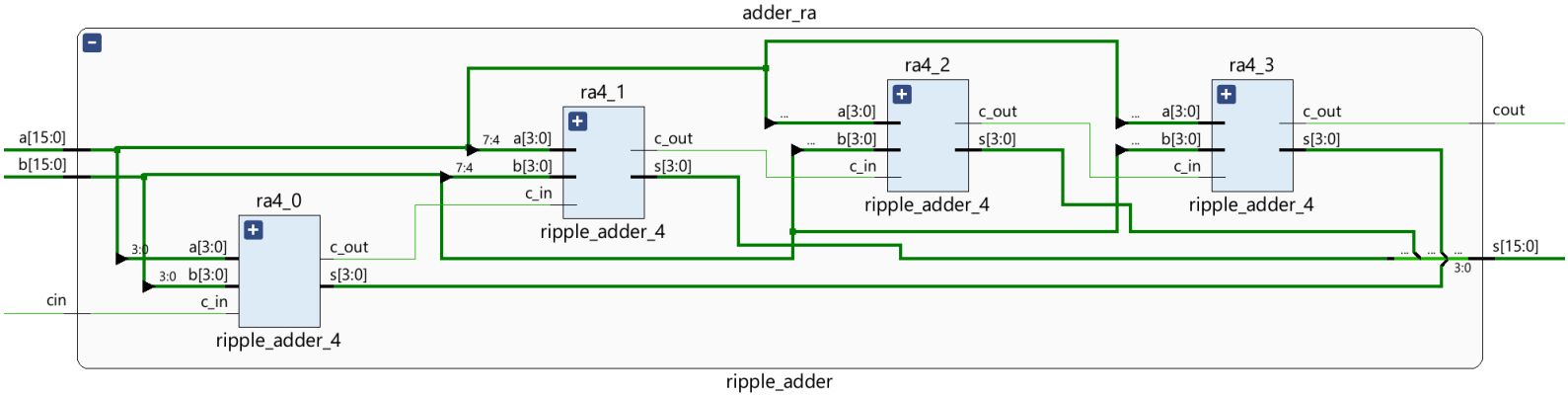

Figure 2: Ripple Carry Adder block diagram from Vivado where each ra4 cell contains 4 full adders.

Carry Lookahead Adder

The carry lookahead adder, as seen in Figure.3, uses the two signals Generated (G) and Propagated (P) to calculate the carry-out of each full adder depending on the available inputs A and B. A carry-out is generated whenever both A and B are 1, no matter what the carry-in is as follows: **G(A,B)=AB**. Similarly, the previously calculated carry-out is propagated if either A or B is 1, **P(A,B)=A^B**. Therefore, the carry-out is calculated by $C_{out\ i+1} = G_i + (P_i\ C_i)$. This way, the carry-ins of each bit don't depend on the carry-out of the previous sum, which makes this design faster than the ripple adder. However, building an N-bit lookahead adder will make the carry-in expressions very complex, using many logic gates. So, in order to make this design work, it should be constructed in a hierarchical fashion. We are using a 4x4 hierarchical design for our 16-bit inputs, as seen in Figure.4. The expressions for the four full adder carry-outs are as follows:

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

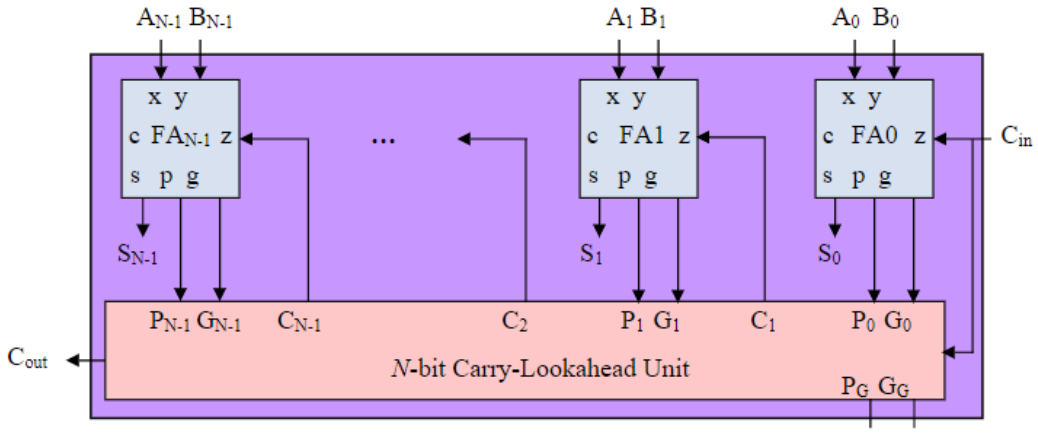

Figure.3: Block diagram of a 4-bit group of the carry-lookahead adder

After that, each 4-bit group will output two signals that will be used to calculate the carry-in for the next 4-bit group propagate (P_G) and group generate (G_G), and their logic is:

$$\mathbf{P_G = P_0\ P_1\ P_2\ P_3}$$

$$\mathbf{G_G = G_3 + G_2\ P_3 + G_1\ P_2\ P_3 + G_0\ P_1\ P_2\ P_3}$$

Denoting the P_G and the G_G from every 4 bits as $P_{G0}\ P_{G4}\ P_{G8}\ P_{G12}$ and $G_{G0}\ G_{G4}\ G_{G8}\ G_{G12}$ to use them in calculating the carry-ins for each 4-bit group, instead of just connecting the fourth bit's carry-out to the next first carry-in, which defeats the purpose of speeding up the process. The expressions for the carry-ins of the 4-bit group are as follows:

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

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

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

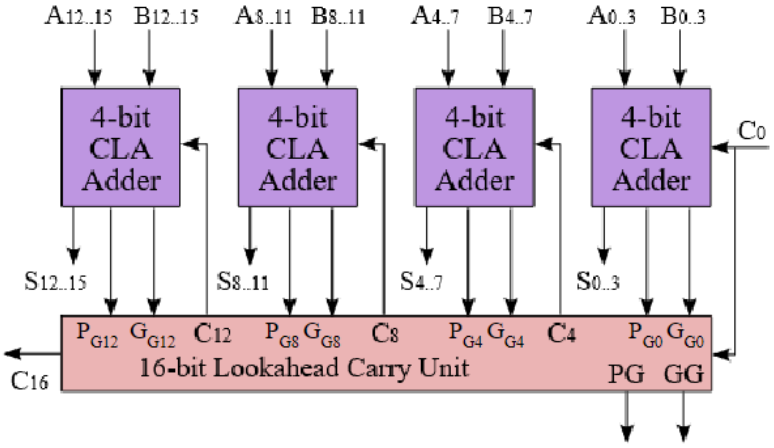

Figure.4: High-level block diagram of hierarchical design of the carry-lookahead adder

Carry Select Adder

Just like the previous adders, we implemented the Carry Select Adder in a hierarchical manner. The first two layers are very similar to the ripple carry adder, with the first layer consisting of a full-adder circuit and the second layer consisting of a group of four full adders tied together, let's call one group of four full adders a Ripple Adder cell (RA).

The third layer is where things get different from the ripple adder. In this third layer, we have 7 RA cells and a few muxes. The first RA cell would have the first four bits of the numbers we want to add and the carry-in connected to it. Furthermore, three RA cells would compute the addition of the remaining 12 bits of the two numbers with the assumption that the carry-in for each RA cell is zero, while the other three RA cells would compute the exact same addition. However, the assumption here is that there is one carry-in for each RA cell. Then, the output of each pair of RA is connected to a 2:1 mux. Moreover, the carry-out of each pair of RA cells is connected to a 2:1 mux to determine the select signal for the 2:1 mux that has the output of each RA cell connected to it, as shown in Figure.5.

The reason this approach is way faster than the Ripple Carry Adder, for example, is that once the first RA cell gets the carry-in, the whole 16-bit addition is available after a fixed amount of delays. This is because we compute the addition of the two 16-bit numbers A and B in a parallel manner, meaning that each RA cell does not need to wait for the previous RA cell's carry-out because it is assuming that the carry-in is either zero or one, corresponding to the top and bottom RA groups respectively as shown in Figure 5. The fixed delay mentioned earlier comes from the RA cell itself; since the RA cell consists of 4 full adders connected to each other, each full adder would need to wait for the carry-out of the previous full adder in order to compute the addition, except the first full adder cell.

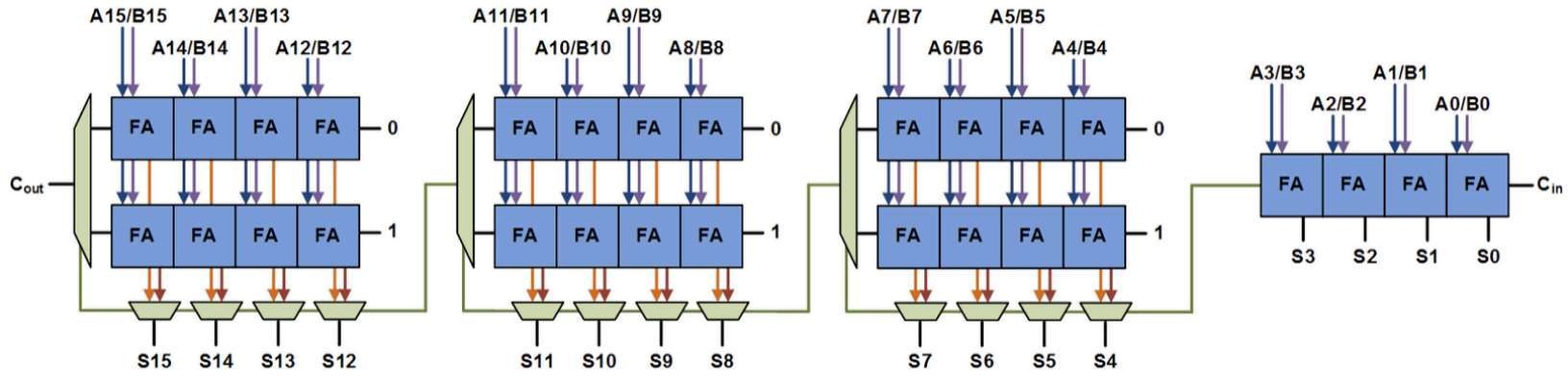

Figure 5: Carry Select Adder block diagram[1]

[1] https://en.wikipedia.org/wiki/Carry-select_adder

Area and Tradeoffs:

- **Carry Ripple Adder:**

The ripple adder is the simplest and most straightforward adder to implement. It doesn't require much logic compared to other implementations. However, this adder has a high delay because to implement N bits, you would have to wait for N full adder delays since each full adder waits for the previous one to finish computation to take the carry-in.

- **Carry Lookahead Adder:**

The lookahead adder has the fastest implementation compared to the other three adders because it predicts the carry-ins without waiting for previous full adders. However, this fast implementation comes with a relatively more complex design with much more logic, which impacts the debugging and building costs.

- **Carry Select Adder:**

The carry select adder is the middle ground between the previous two adders. It has a fast implementation compared to the ripple adder because it computes the sum with the carry-in as 1 and 0 and then chooses the actual value. Moreover, this method has a simpler implementation compared to the lookahead but not as fast.

Conclusion

In this lab, we implemented and analyzed three 16-bit adder designs: Carry Ripple Adder (CRA), Carry Lookahead Adder (CLA), and Carry Select Adder (CSA). Each design had distinct trade-offs: the CRA was simple and resource-efficient but slow due to sequential carry propagation, the CLA was the fastest by predicting carry-ins in parallel but required more logic complexity and resources, and the CSA provided a balance between speed and complexity by precomputing sums for both carry-in values. Performance analysis confirmed that the CLA achieved the highest speed, while the CRA consumed more power due to slower operation.