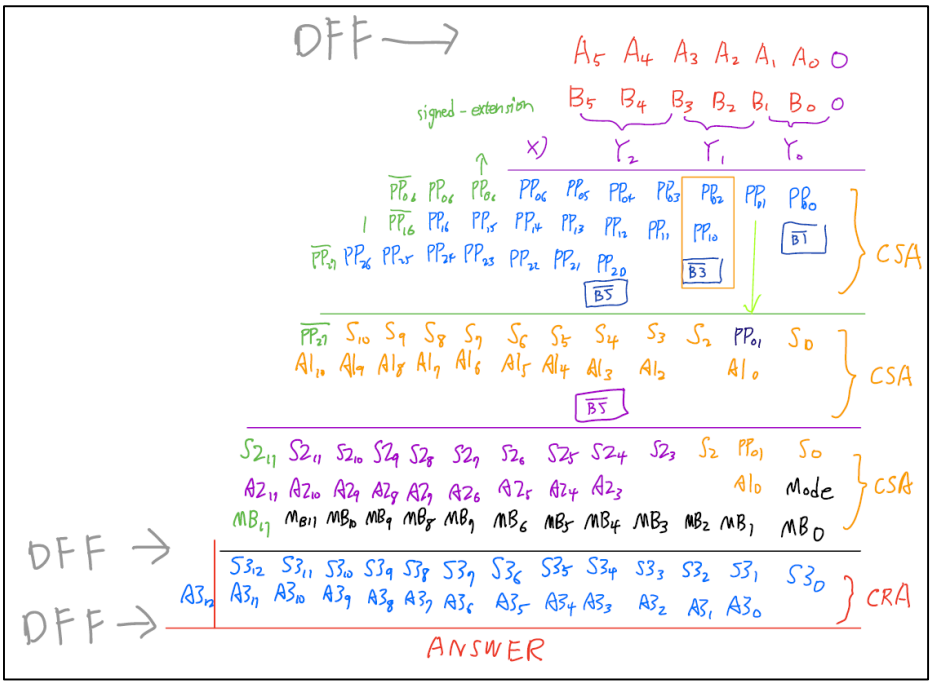
2020 VLSI Final Project

0710130 Ding-Hsiang Huang, 0710145 Jo-Hsuan Hong

I. Summary of structure

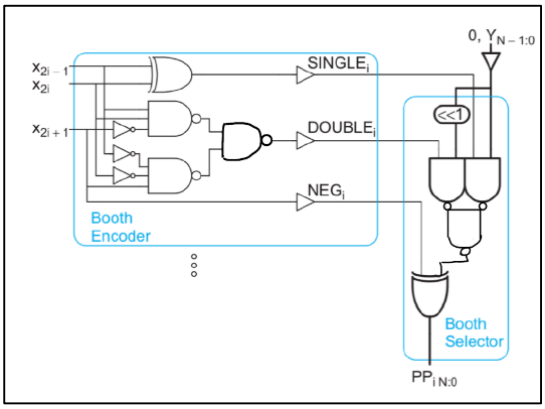
The calculation is according to the Wallace Tree as shown below.



This structure is similar to our midterm project, so we basically didn't change a lot. Utilize the same radix 4 booth encoder to get the partial product, and implement it with CSA, and CRA.

We use three groups of D Flip-Flop in total to divide the whole circuit into two parts. Firstly, input pass through the first stage of D Flip-Flop. According to our calculation, the timing to pass CRA would be greatly larger than the timing of the three stages of CSA plus booth encoding and selection. Therefore, we cut the pipeline before CRA. Finally, after passing through the last stage of DFF, we would get the output we want.

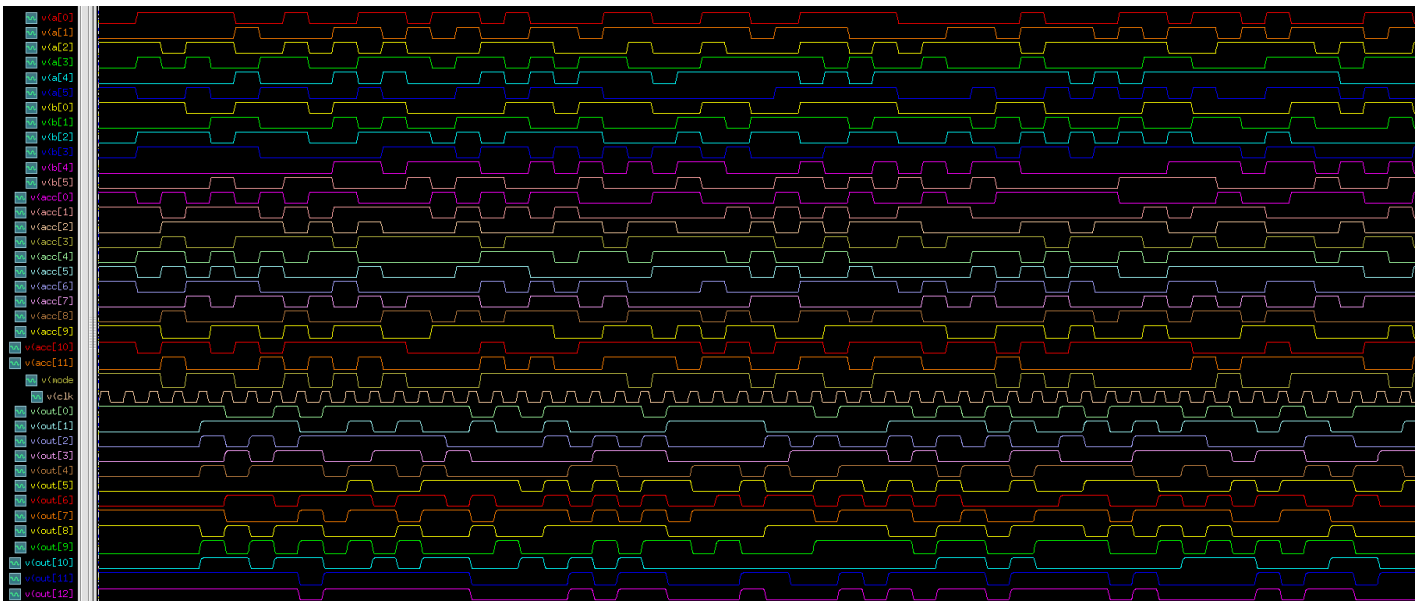
The implementation of **Booth Algorithm** is shown as the figure on the right. Firstly, build each logic gate with pseudo-NMOS, and combine then into encoder and selector, respectively. Using one encoder and seven selectors to build a single component of partial product. And repeat this process for three times, building three components of partial product.



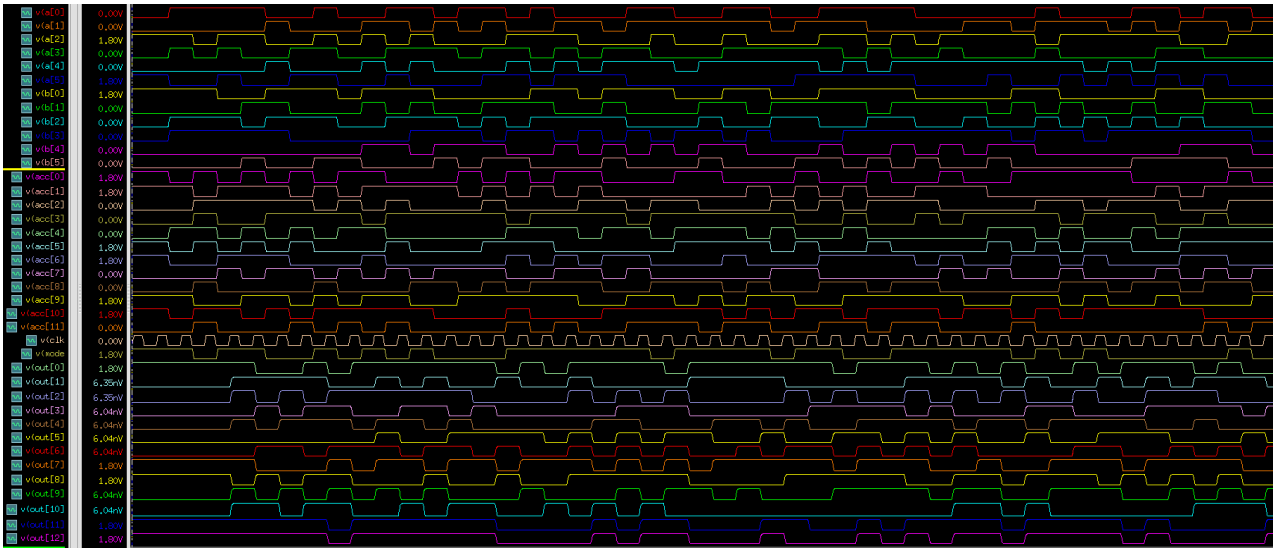
In the Wallace tree, “MB” is a special signal. It is the result of ACC XOR mode. If ACC is positive, there wouldn’t be any difference. If it is negative, we would get its 1’s complement, adding MODE=1, the result signal would become 2’s complement, which is able to continue addition. With this kind of design, it made the last stage CRA can do addition and subtraction at the same time.

II. Output Waveform

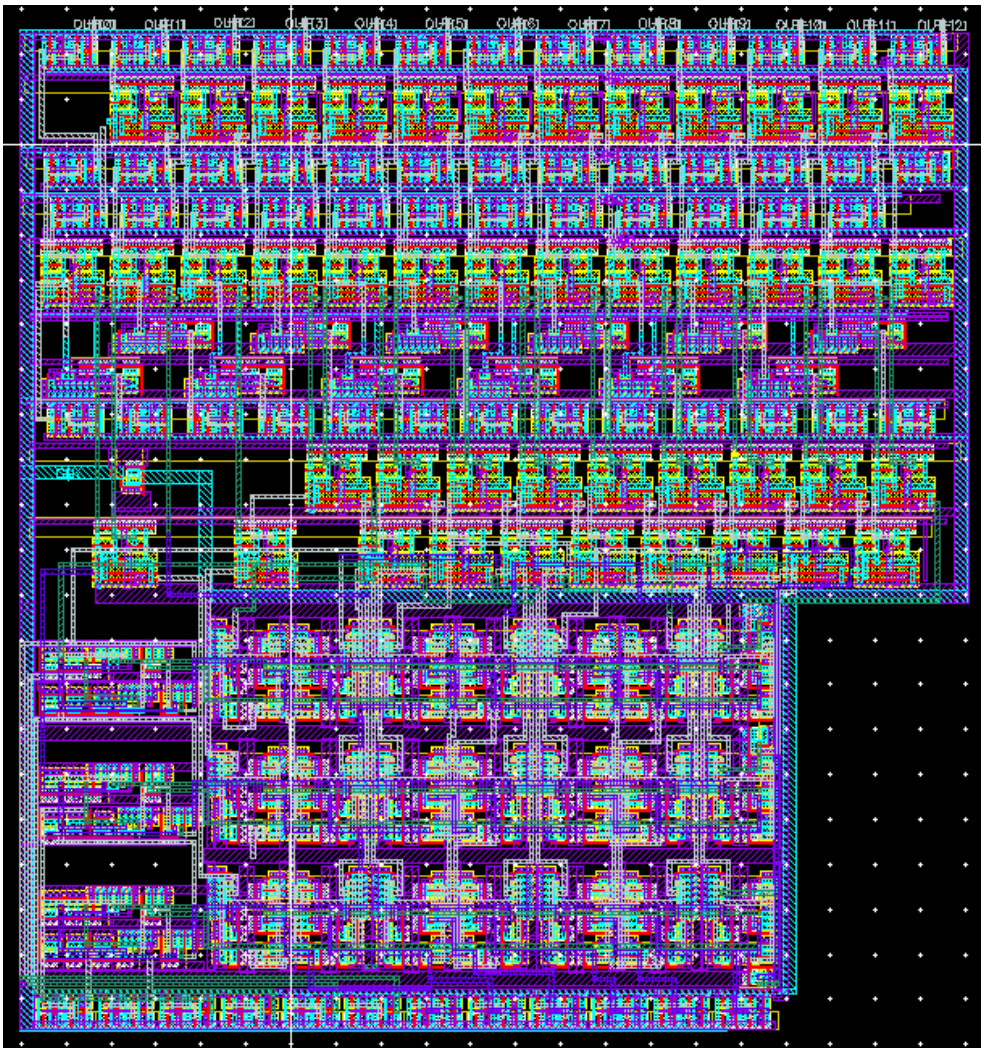
1. Pre-sim waveform:



2. Post-sim waveform



III. Whole Layout



IV. Performance List

Maximum operation frequency	Pre-sim: 1.02GHz (period = 0.98ns)
	Post-sim:769MHz (period =1.3 ns)
Average power	Pre-sim: 102.3945m
	Post-sim:102 0086.m
Layout area	105.53u * 111.25u = 11740.2125u^2
Multiplier and adder structure	Booth Algorithm and Carry-Ripple Adder

V. Discussion

A. Pre-sim

Since this assignment is based on the midterm project,the pre-sim process would be converting the original Verilog code into an SP file. The only change we made is to replace the full adder with the compound to reduce the area. Moreover, we massively utilize pseudo-NMOS to accelerate the speed. The greatest obstacle we confronted in pre-sim is the adjustment of the transistor's width. Since some of them are pseudo-NMOS while others are compound, we spent a long time precisely adjusting widths. However, it was weird that after converting the circuit into an SP file, replacing CLA with CRA resulted in a faster speed of the circuit. Moreover, the structure of CRA is simpler, so we decided to use CRA in implementation.

Same as the midterm project, there are two modes, addition and subtraction, we manage it by letting ACC perform XOR with the mode first, and add the mode again. In this method, if the mode is "1", ACC would be reversed, and adding the mode. And it would become 2's compliment. At last, we could simply let it add A*B and get the final result.

B. LAYOUT

In order to the convenience of connecting different circuits, not only setting proper specs for commonly used wired, VDD, GND, and CLK but also reserving areas for the pin and via. Preventing the annoyance of adjusting each component respectively after its number grows.

Besides, our work division is one person starting from the input port's DFF and booth algorithm, and the other person starts from the output port's DFF and CRA. In this manner, we could divide the whole layout into several smaller cells and pass the DRC and LVS beforehand, and even debugging respectively. Later, we could directly call an instance when we need to use it in the bigger cell, which saves lots of time and effort.

For drawing layout this time, we've tried multiple built-in tools of Virtuoso. We mirrored or rotated when we were connecting cells, in order to share GND or VDD and reduce area usage. Also, we used the view tool on the left to help filter out the desired item respectively, and increase our debug speed. After all, the efficiency is better than the last several times.

Originally, we used half adder in the SP file, but using XOR gate and AND gate had worse performance in both time and area than compound Full Adder made by pseudo-NMOS. Hence, we replaced all of them with Full adder.