ECE 260A – Lab 3

4-Tap 16-Bit Unsigned Averaging FIR Filter

Feiyu Yang PID: A59009686

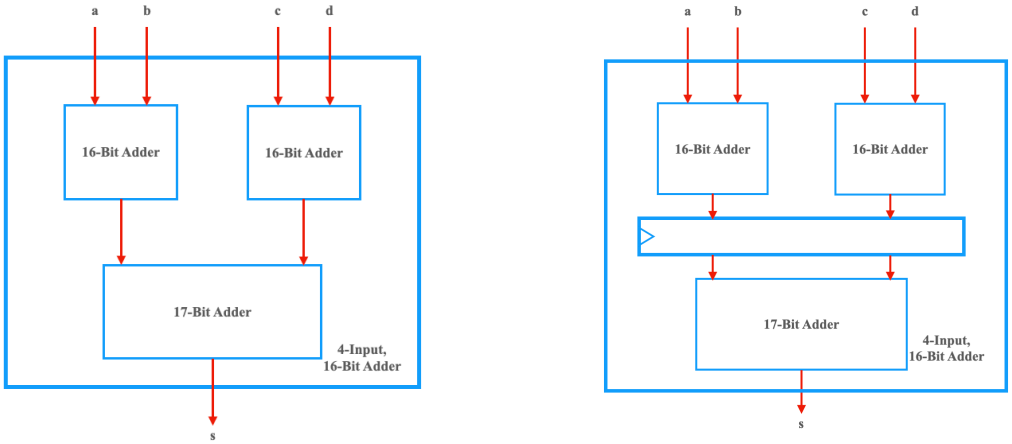
Design Methodology

Top Module (FIR_pipeline.sv)

Although the formality of this design is a FIR filter, since the kernel of this convolution is a constant number 1, the main function is a 4-input, 16-bit adder. The 4 inputs of this adder are the outputs of a series DFFs which capture the input 16-bit signal.

Assume adder is provided. Since there are 4 inputs and most of the adders only have 2 inputs, one of a natural thought is to use a 16-bit adder to compute inputs a and b, which produces a 17-bit result s1, and an identical 16-bit adder to compute inputs c and d and get s2. After that, a 17-bit adder is used to compute the sum of s1 and s2, and eventually get the 18-bit result s. The structure is illustrated in Figure 1.1 (a).

With this tree structure, an obvious optimization is to add a pipeline before the third adder. In this way we reduce the path the data needs to travel through in one clock cycle and thus gain potential benefits. This structure is shown in Figure 1.1 (b).



(a) (b)

Figure 1.1 High Level Implementation of 4-input 16-bit Adder

Carry Select Adder

The reference project doesn't do good in terms of data arrival time. One of the main reasons is the nature of ripple carry adder. There's a strong dependance between one full adder's output s, c_out; and the input c_in, which is last full adder's output. One adder needs to wait the previous adder finishing the computation before starts to work, therefore causes a longer delay.

Other than pipelining, another idea to cut down this dependance and thus reduce the propagation delay is to use carry select adder. Instead of using last adder's output c_out to compute the result s and c_out, carry select adder contains two adders with c_in to be 0 and 1 respectively, and uses the actual c_in as the select signal for the multiplexor to choose the correct output. In this way there's little dependency between the current adder the previous adder. But there're two downsides: It introduces MUX which will cause extra delay, and it almost doubles the number of gates, resulting the significant increase of area and power consumption.

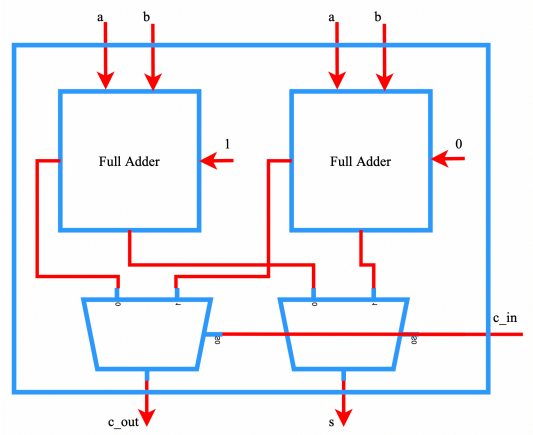


Figure 1.2 Carry Select Adder

When designing N-bit carry select adder, it's not the best idea to form it with N 1-bit carry select adder since that way the main delay will come from N MUXs. Generally it is considered a good idea to use $\sqrt{N}$ stages with each stage formed by a $\sqrt{N}$-bit ripple carry adder and 2 MUXs. In this design, 4-bit-adder.sv provides basic 4-bit ripple carry adder. Two such modules and two MUXs are used to form a 4-bit carry select adder, 4-bit-adder-select.sv. The 16-bit adder, 16-bit-adder-select.sv is formed with 4 such adders. 17-bit adder, 17-bit-adder-select.sv is constructed in a similar fashion, with the MSB calculated by an additional 1-bit full adder.

This concludes the first design choice. The code can be found at https://github.com/LePetitPrince612/ECE260A under Carry-select folder. Next I'll introduce a more radical design which will reduce the delay even further.

Pipelined 16/17-Bit Adder

When looking into the 16-bit adder, although each sub-module uses carry select adder, there's dependance between each 4-bit adder. Therefore, the delay could be optimized by introducing pipelines between these 4-bit adders.

This design adds a pipeline between every two 4-bit adders, resulting a total of 7 pipelines in the circuit. Although this will reduce the delay, more power consumption and larger area is foreseeable since a lot of DFFs are involved. Related code can be found at https://github.com/LePetitPrince612/ECE260A under Carry-select-pipeline folder.

Synthesis Result

Timing

	Reference	Carry Select Adder	Carry Select with Pipelines
Library Setup Time	-0.03	-0.03	-0.12
Data Required Time	0.47	0.47	0.38
Data Arrival Time	-0.65	-0.51	-0.38
Slack	-0.18	-0.04	0.00

Area

	Reference	Carry Select Adder	Carry Select with Pipelines
#Combinational Cell	826	419	937
Combinational Area	1820.520030	876.600016	1621.080042
#Sequential Cell	82	116	347
Non-combinational Area	636.839990	830.520002	2517.119987
#Buffer Cell	4	91	0
Buffer Area	5.76	4.32	0.00
#Inverter Cell	240	88	184
Inverter Area	312.12	106.20	202.68
Total Cell Area	2457.360019	1707.120018	4138.200029

Power Consumption

	Reference	Carry Select Adder	Carry Select with Pipelines
Internal Power	1.4653mW	1.6868mW	4.9991mW
Register	1.2506	1.6024	4.8573
Combinational	0.2147	0.084384	0.1418
Switching Power	0.3517mW	0.1243mW	0.2172mW
Register	0.1579	0.069668	0.1124
Combinational	0.1928	0.054644	0.1048

	Reference	Carry Select Adder	Carry Select with Pipelines
Leakage Power	25.1938uW	14.6113uW	30.0823uW
Register	17.863	7.8031	19.698
Combinational	7.3300	6.8079	10.385
Total	1.8422mW	1.8257mW	5.2464mW
Register	1.4158	1.6798	4.9894
Combinational	0.4264	0.1458	0.2570

Analysis and Conclusion

In timing report, both design have significant improvement comparing to vanilla ripple carry adder design. For Carry Select Adder design, this is due to the nature of carry select adder and the added 1 pipeline. The second design is heavily pipelined and shows better timing result.

In area report, Carry Select Adder design added extra pipeline, therefore have more sequential cell number and area. Heavily pipelined second design involves even more of them. But weirdly the first design also significantly reduced combinational logic number and area. After check the generated schematic (Figure 3.1), everything are generated as expected, so the reason for that remains uncertain.

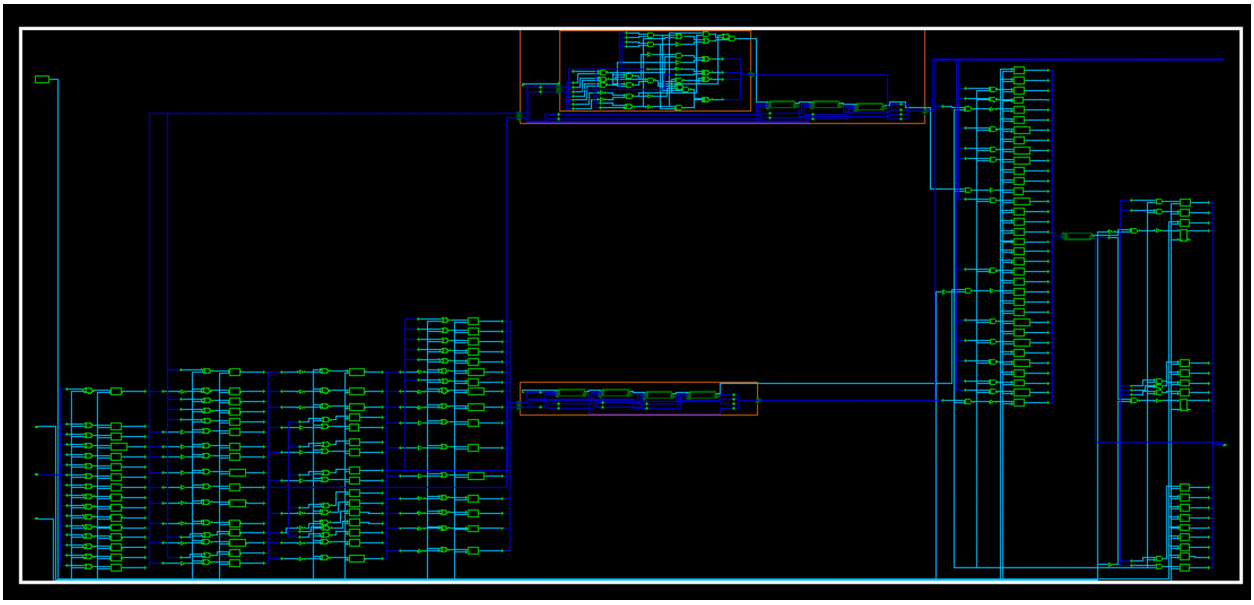


Figure 3.1 Carry Select Adder Schematic

In power report, Carry Select Adder design greatly reduced power assumption of combinational logic, considering in area report it only contains about half of the combinational logic comparing to reference design. The second design used a lot more registers, and hence has more power assumption.

In conclusion, both design realized the goal to optimize timing performance. In the second design we could also see the tradeoff between timing and size/power.

Potential Further Improvements

Due to the limited time I had working on this project, I didn't have chance to actually test the delay of a full adder and a MUX. If such data is provided, the 16/17-bit adder may be further optimized. For example, if MUX and full adder have similar delay, by assigning more bits to later stage since they need to wait longer to get the sel signal, the total path delay could be reduced even more.

Besides, the second design focused on dealing with the dependency between 4-bit ripple carry adders, which essentially is the delay caused by MUXs. In stead of trying to use pipeline to reduce this delay, it may have a better effect if pipelines are put within the ripply carry adders.