EE476 VLSI Final Project: Accumulator

Yuxiang Chen, Xinyi Chang, CheeKai Tan

Abstract — These report gives a guideline to build a 16-bit sign-magnitude data accumulator, which output a 21-bit data every 64 cycles as it continually processing the incoming data. Along with the design specification, implementation, optimization and testing strategy will also be analyzed in this report.

Introduction

This report serves as a guideline on how to build a 16-bit sign-magnitude input accumulator module, which output 21-bit data every 64 cycles. The module has three inputs: clock, 16-bit sign-magnitude input, reset and one 21-bit 2's complement output. As mentioned in the functional specification, data will arrive every cycle, but the output changes only update lastest sum-of-64 computation as a 2's complement. As part of the project, both schematic and post-layout are implemented. A variety of testing and optimization are also accomplished in this design. The design is sized to drive a 20fF load, and inputs are assumed to be provided by a buffer of drive 0.5/0.2 um.

The project is implemented in a group of three people. And the project is saved in Yuxiang Chen's directory, named 'project'.

Implementation

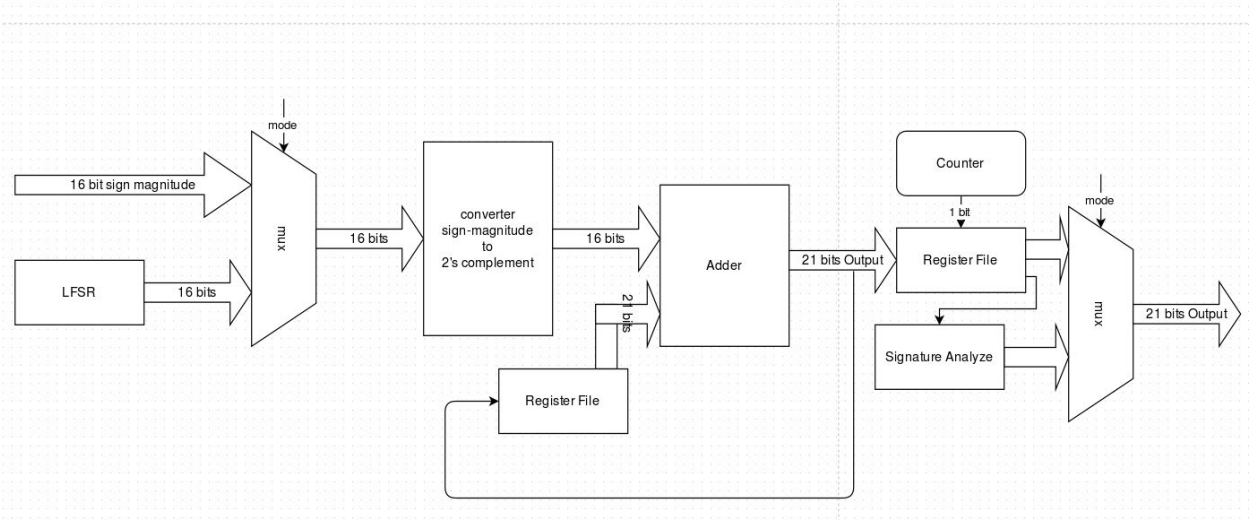

Figure 1.1 Overall Block Diagram

To design the module which meets the specification, we first draw a high-level block diagram as shown in Figure 1.1. The design has been divided into three stages: input stage, accumulate stage and output stage.

Input Stage: the data can be generated by user or LFSR, both are providing a 16 bit sign magnitude data, and using a 16 bit mux array to select which input method. If mode 1 is selected, user input will go through, if mode 0 is selected, LFSR input will go through.

Accumulate Stage: after inputting the data, the 16-bit sign magnitude data will first go through a converter, which will convert the sign magnitude data to 2's complement. This can be implemented by XOR all the bits with the MSB. And the output will be one of the input for the adder. Another input will be provided by the register file 1, which is feeded by the output of the adder. The register file 1 is triggered by rising edge of the clock. By this way, the adder will do addition with the previous data every clock cycle as it continually processing the incoming data.

Output Stage: the output stage is implemented by a counter, a register file, a signature analyzer, and a mux array. We first implemented a 6 bit synchronous counter, which counts from 0 to 63. The counter will output a 1 at the rising edge of the next clock cycle of count 63. In this way, we can use this signal to trigger the register file 2 to output sum every 64 cycles. This signal is also used to reset register file 1 after each 64 cycles. Last but not least, the output of register file can be directly output or go through a signature analyzer to test functionalities.

Adder Design: Sklansky's Adder

To design an adder that is fast but also easy to implement, we choose to use a 21 bit tree adder, sklansky adder. Although it suffers from fan-out problems, the architecture is simple and regular. Figure 2.1 shows the design of our adder.

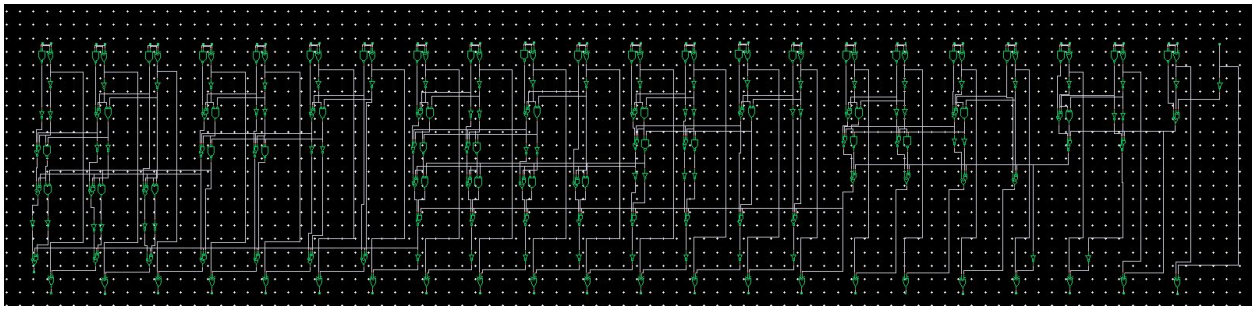

Figure 1.2 Sklansky's Adder Schematic

Linear Feedback Shift Register

To test the functionalities of our design, we implemented a 16 bit galois-type LFSR with the polynomial of 1+x^4+x^13+x^15+x^16. This will generate pseudo-random data.

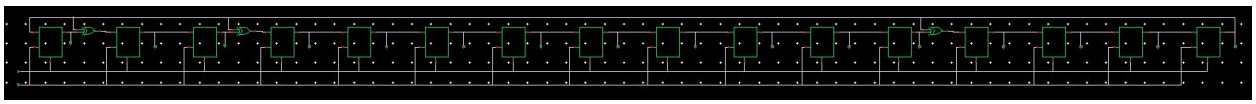

Figure 1.3 LFSR Schematic

Synchronous Counter

To design a counter that can count 64 bit and synchronous with the clock, we implemented a 6-bit synchronous counter. when all six bits are 0s, it will output a 1 to trigger the reset of register file 1 and clock of register file 2.

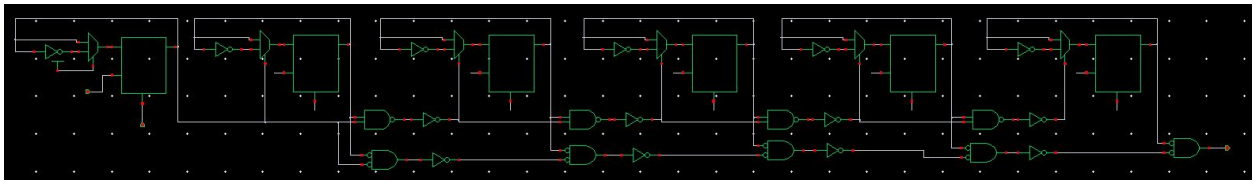

Figure 1.4 6-bit Synchronous Counter

Results

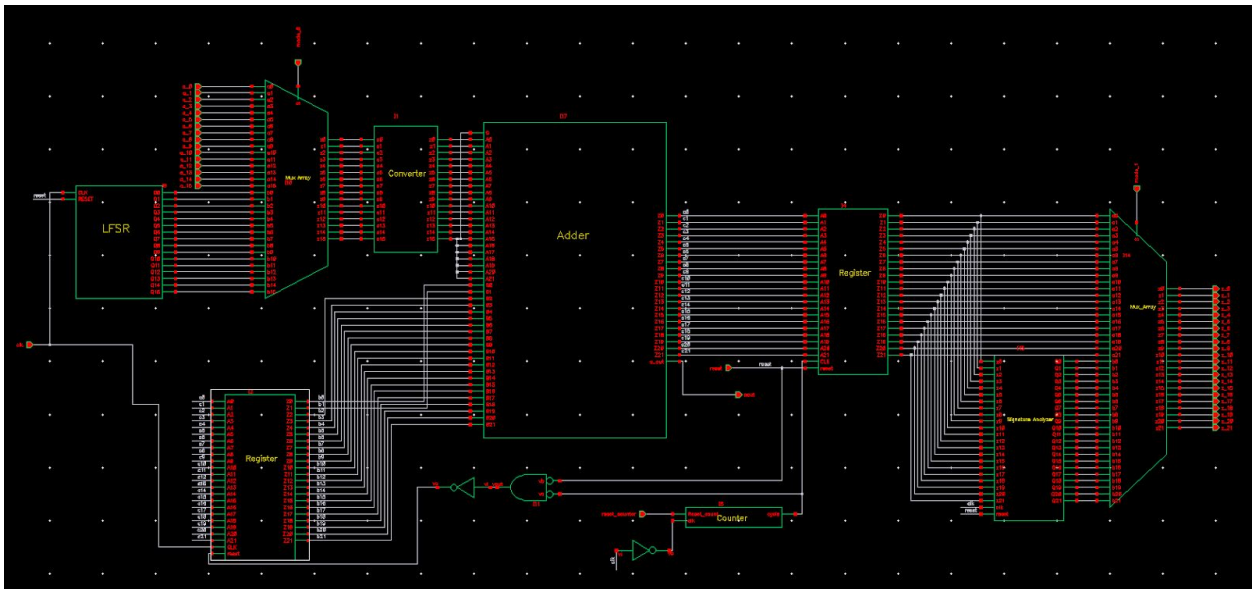

Figure 2.1 Schematic Screenshot

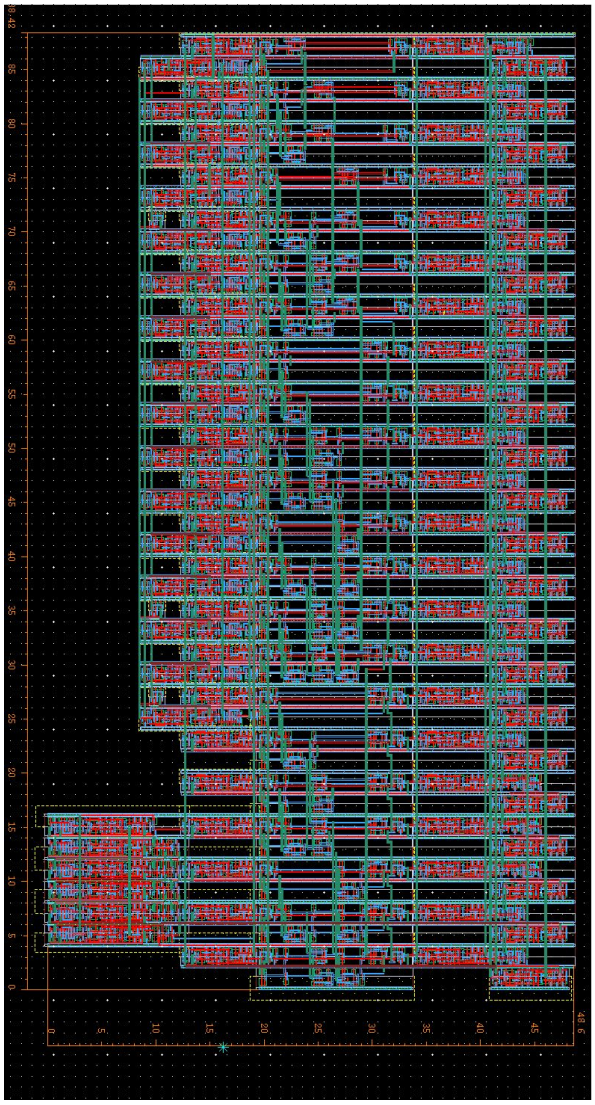

Figure 2.2 Post Layout Screenshot

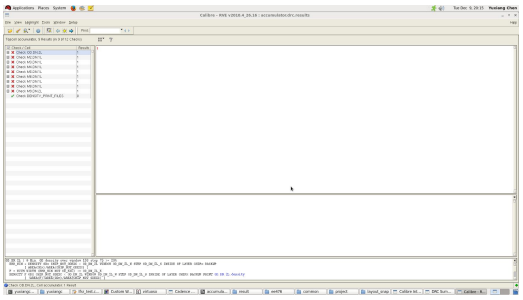

Figure 2.3 DRC Clean

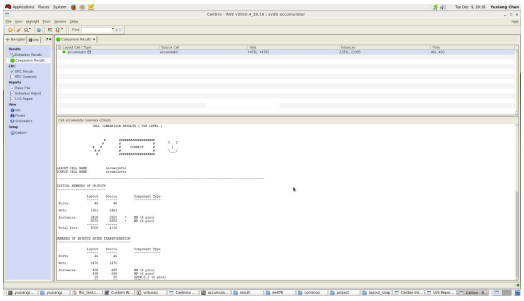

Figure 2.4 LVS Clean

Our accumulator was designed as in Figure 2.1 and Figure 2.2. The layout of our accumulator has passed both DRC and LVS (DRC density error is negligible).

Schematic Simulation:

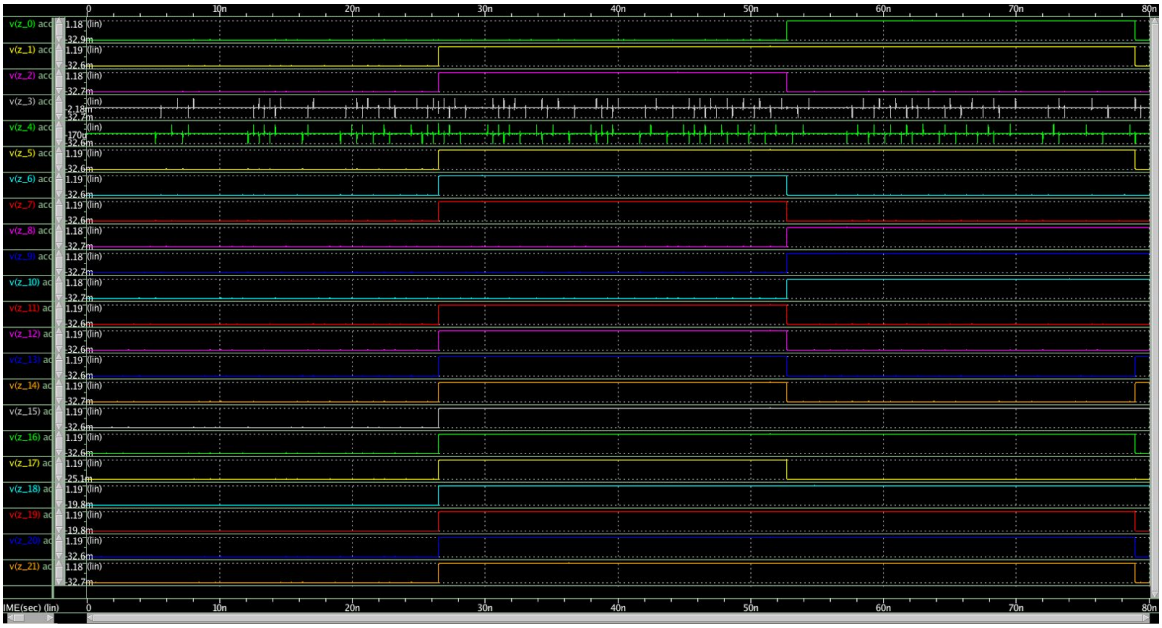

Figure 2.5 Schematic Simulation Waveform using Provided Testing Input

The functionality of the schematic of our accumulator was tested using the given input.ctl file. As shown in Figure 2.5, the result of the simulation is the same as the given answer (first 64 cycle: b'1111111111100011100110; second 64 cycle: b'1111011000011100100011), which proves that the schematic of our accumulator has functioned properly.

Energy Dissipation:

Figure 2.6 Schematic Simulation Waveform using LFSR Input

In addition to manual input, we have also tested our accumulator by using LFSR input. Using Matlab, we found out the sequence of our LFSR (which begins with b'00000000000000000001) and the sum of 64 sequences. The result of the simulation is illustrated in Figure 2.6, which matches the answer generated by Matlab
(first 64 cycle: b'00000100000010110000000; second 64 cycle: b'11111101001000011011010).

To find out the critical path of our accumulator, we used the input transition of (b'00000000000000000000 -> b'10000000000000000001 -> b'00000000000000000001 -> b'00000000000000000000), which basically means adding -1 to 1. The result of these inputs should be 0, which we have accomplished. Using this critical input, we tried to reduce cycle time to get the minimum cycle time that would still give us the same result.
As a result, the **minimum cycle time** of the **schematic** of our accumulator using 1.2V is **430ps**.
Total Computation Time: 28 ns
Energy Dissipation(schematic): 71.8p J

Post Layout Simulation:

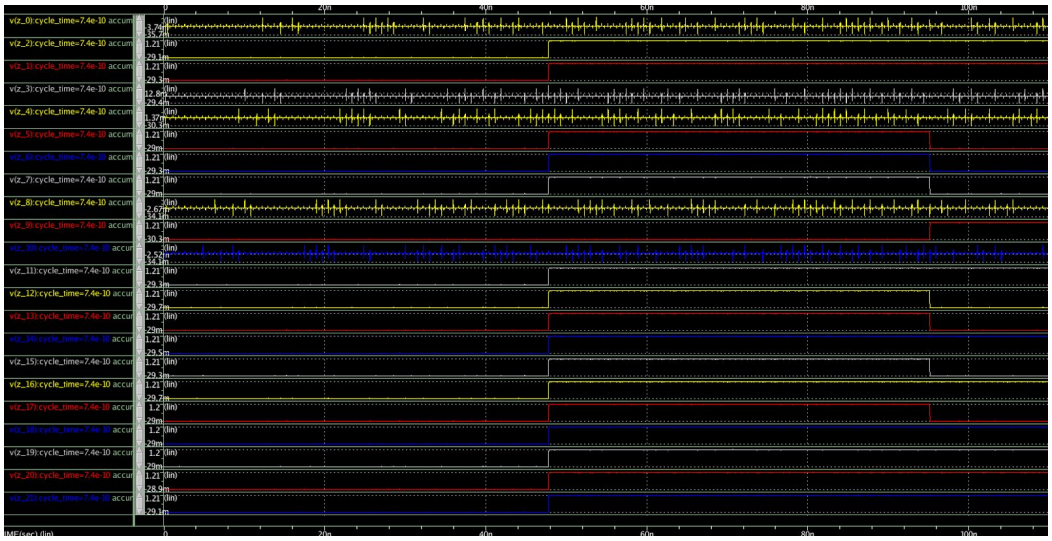

Figure 2.7 Layout Simulation Waveform using Manual Input

Using the same manual input stated in schematic simulation, we tested the layout of our accumulator, and the result was correct as shown in Figure 2.7. Furthermore, using the same critical input, we found out the **minimum cycle time** of the **layout** of our accumulator using 1.2V to be **730ps**.
Total Computation Time: 47 ns
Energy Dissipation(layout): 127p J

Conclusion

In summary, our design and implementation meet the specification and passed functionality test. We achieved a minimal cycle time of 430 ps for schematic and 730 ps cycle time for post-layout. Our post-layout is compact and well-organized, but the counter layout could be implemented more consistent

with the overall blocks to shrink down the area further more. Also, due to time limit, the transistor size isn't fully optimized. This could also be improved in the future.

Appendix

All files are located at /homes/projects/ee476/yuxiangc/project/result

Schematic Simulation w/ Manual Input:

Netlist:	accumulator_schematic.ckt
Control file:	accumulator_schematic.ctl
Simulation file:	accumulator_schematic.tr0
Waveform Pic:	accumulator_schematic.png
Answer pdf:	accumulator_schematic_answer.pdf

Schematic Simulation w/ LFSR Input:

Netlist:	accumulator_schematic.ckt
Control file:	accumulator_lfsr.ctl
Simulation file:	accumulator_lfsr.tr0
Waveform Pic:	accumulator_lfsr.png
Answer text:	accumulator_lfsr_first64.txt (answer for the first 64 cycles)
	accumulator_lfsr_second64.txt (answer for the second 64 cycles)

Layout Simulation:

Netlist:	accumulator_layout.ckt
Control file:	accumulator_layout.ctl
Simulation file:	accumulator_layout.tr0
Answer pdf:	accumulator_schematic_answer.pdf
Extaction files:	accumulator_layout.pex.netlist
	accumulator_layout.pex.netlist.ACCUMULATOR_LAYOUT.pxi
DRC summary:	drc
LVS report:	lvs

Snapshots:

Block Diagram:	block diagram.png
Schematic:	schematic.png
Layout:	layout.png
Layout DRC Clean:	drc_clean.png
Layout LVS Clean:	lvs_clean.png