WonBin Seo

Dr. Shannon Nicley,

ECE 813 Advanced VLSI Design

7 April 2025

8 bit Microprocessor Design with Cadence

ECE 813 Design Project Proposal

Michigan State University

1. Project Overview

This project designs the schematic and layout of an 8-bit microprocessor data path, with an ALU, a barrel shifter, and a register file using Cadence Virtuoso design tools.

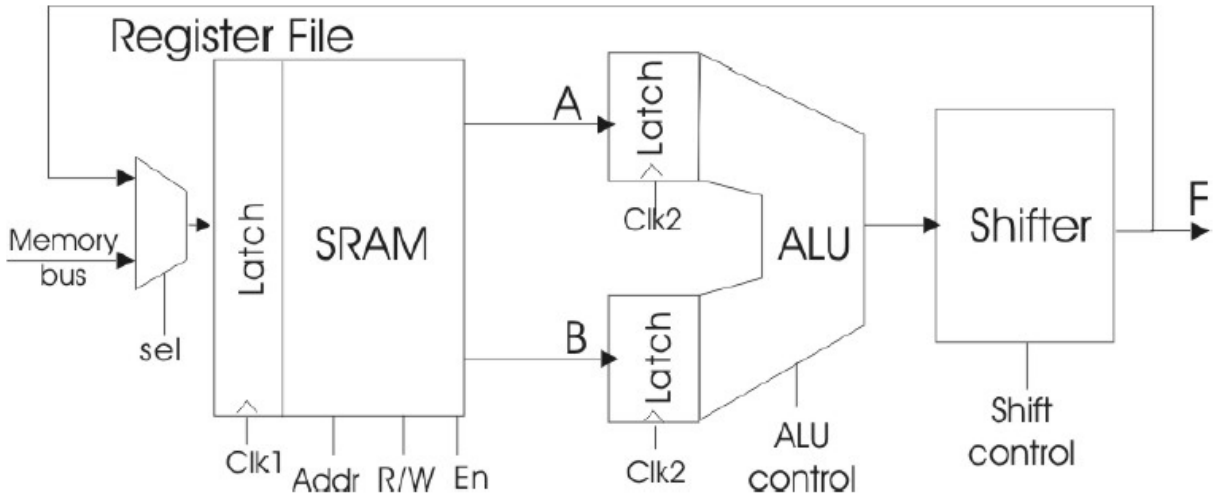

Figure 1. Structure of the 8-bit microprocessor data path

In this data path, we must design a complete 8-bit data path consisting of a register file (SRAM), an ALU (arithmetic logic unit) and a shifter, as shown in Figure 1. The data path will be capable of multiple instructions determined by several control signals that comprise the operational code or “opcode”.

The ALU can perform both arithmetic and bit-wise logical operations. The ALU will have two 8-bit data inputs and several control signals that specify the ALU function to be executed in a given clock cycle. The ALU input data is loaded into the latch at the rising edge of the ALU clock (clk2), and the output is generated after a delay due propagation through the combinational logic of the ALU and the shifter. ALU will contain add/subtract and other logical calculations, using Manchester carry generation carry lookahead adder.

A shifter provides data manipulation capabilities. In this design, we will use barrel shifter that executing logical shifts.

The register file is a block of memory that provides the data inputs to and stores outputs from the ALU. The register file at read/write address is specified by control bits included in the opcode. We will use multiport SRAM, has two read line and one write line.

2. Summary of Design Challenges

In this project, we will confront many types of challenges. Firstly, the whole data path is compromised in limited area. Using the design specifications which are offered by the Cadence, we will design the path as small as possible. For this reason, we must find which component has the largest portion of the design, trying to reduce the amount of the area. Secondly, Divide the design into small pieces. For example, in 8-bit design, we can divide this design into 4-bit, 2-bit, and 1-bit. Applying this characteristic, we can implement the small design and easily repeat it. In the design of the ALU, Register File, and Shifter, we will use it effectively.

3. Work Plan

i. ALU

Design the ALU that consists of the adder/subtractor, logical calculator, and increment/decrement.

When designing adder/subtractor, we must use Manchester Carry Generation Adder and carry lookahead PG block.

For the Manchester Carry Generation Adder, we will design 1-bit adder and expand to 4-bit adder. Similarly, we will design Carry Lookahead PG block in 4-bit and combine these two as 8-bit the Manchester Carry Generation Carry Lookahead Adder. We must notice that Carry Lookahead PG block cannot be designed as 1-bit block because it needs at least 4-bit when predicting the carry.

Then, we need 8-bit logical calculations to complete the ALU.

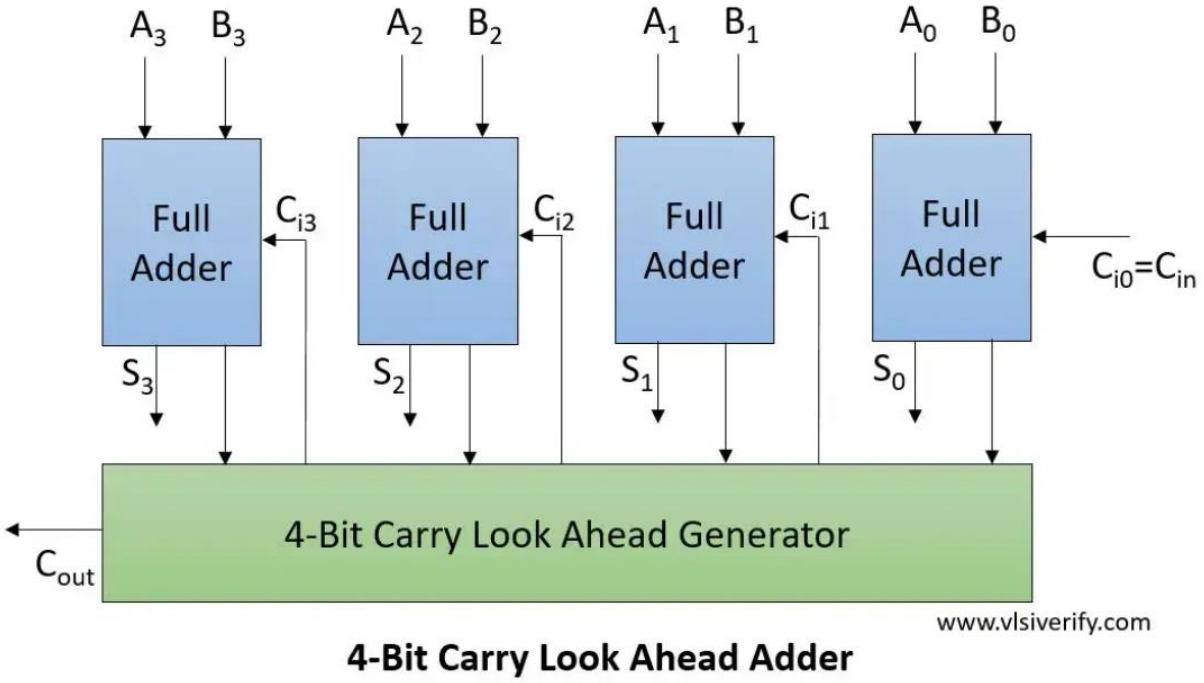

Figure 2. 8-bit Manchester Carry Generation Carry Lookahead Adder (Full adder =

Manchester Carry Adder)

A	B	A+B	A-B	Increment A	Increment B	A NAND B	A NOR B	A XOR B	NOT A
0	0	0	0	1	1	1	1	0	1
0	1	1	(-)1	1	0*	1	0	1	1
1	0	1	1	0*	0*	1	0	1	0
1	1	0*	0	0*	1	0	0	0	0

There is carry out at *

Table 1. 2-bit ALU calculation

A	B	A+B	A-B	Increment A	Increment B
11111111	00000000	11111111	11111111	00000000*	00000001
11111101	00000001	11111110	11111100	11111110	00000010

Table 2. 8-bit ALU calculation examples

A	B	A NAND B	A NOR B	A XOR B	NOT A
11111111	00000000	11111111	00000000	11111111	00000000
11111101	00000001	11111110	00000010	11111100	00000010

Table 3. 8-bit ALU calculation examples

F2	F1	F0	Operation
0	0	0	A+B
0	0	1	A-B
0	1	0	Increment A
0	1	1	Increment B
1	0	0	A NAND B
1	0	1	A NOR B
1	1	0	A XOR B
1	1	1	NOT A

Carry in is substitute by F0, F1, or F2

Table 4. Control bits and operation of ALU

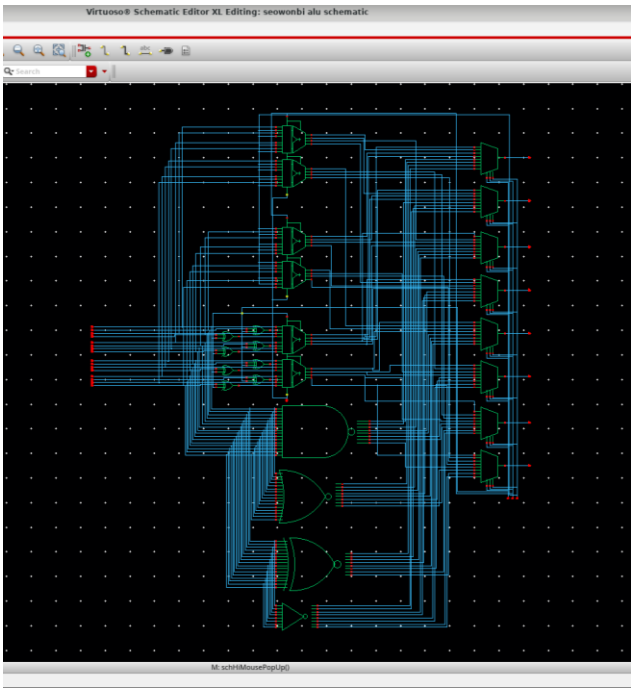

Figure 3. ALU Schematic

ii. Shifter

In this design, we will use the barrel shifter that consists of the Mux 2:1. We can implement the design using Mux 8:1 but use lots of Mux 2:1 to implement logarithmic barrel shifter. This is because it has high speed operation ($O(\log_2 N)$) and efficient hardware implementation. By using it, we can implement logical shifts.

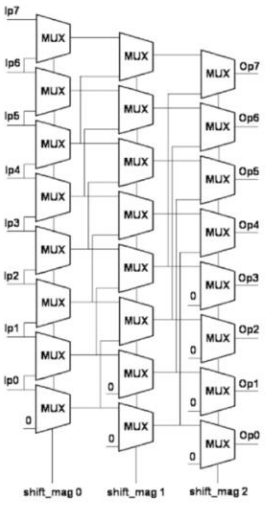

Figure 4. 8-bit barrel shifter

F5	F4	F3	Operation
0	0	0	No Shift
0	0	1	LSR1
0	1	0	LSR2
0	1	1	LSL1
1	0	0	LSL2

Table 5. Control bits and operation of Shifter

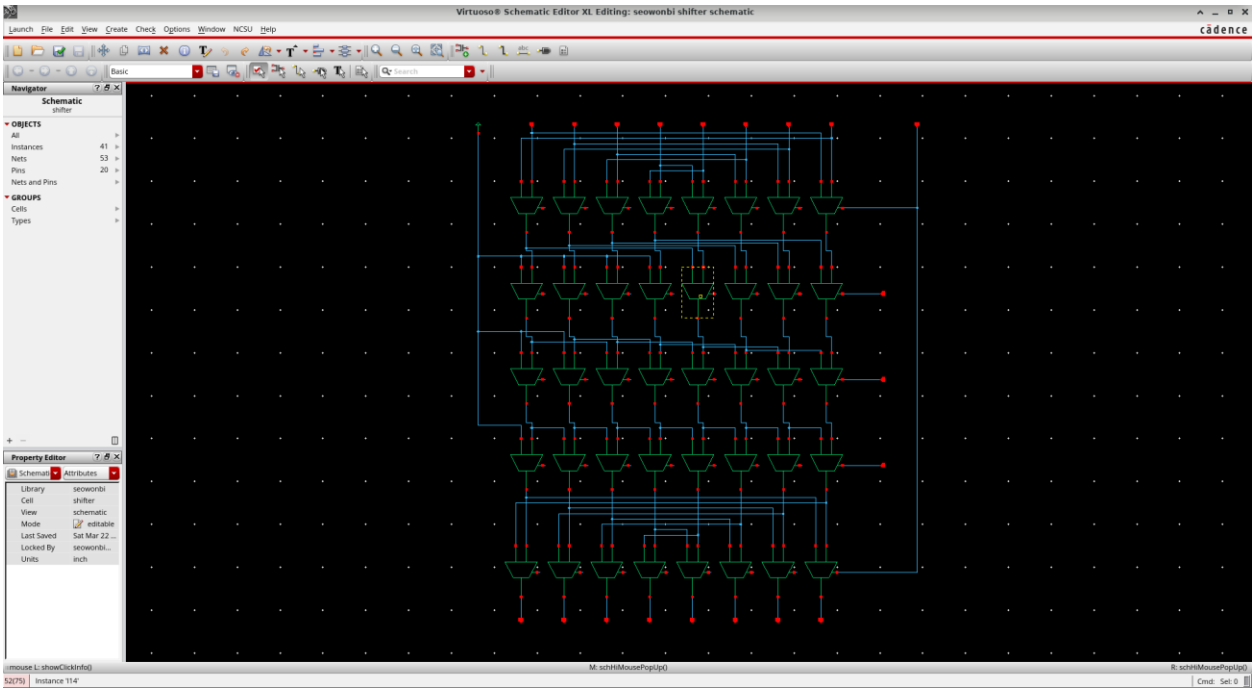

Figure 5. Shifter Schematic

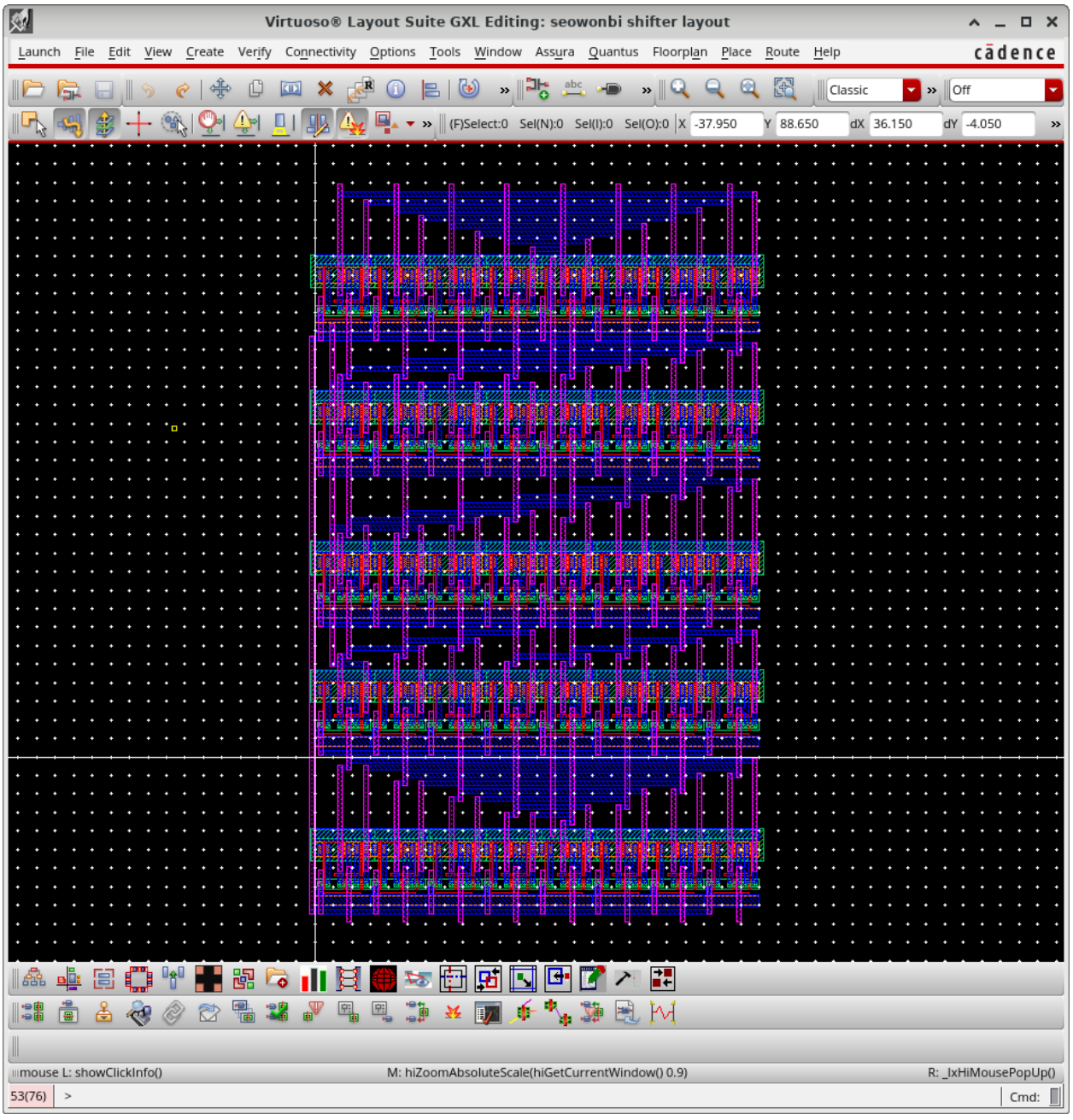

Figure 6. Shifter Layout

iii. Register File

We will use multiport SRAM that has two read lines (outputs), and one write line (input) to implement 8 × 8 SRAM Register File. This will consist of a SRAM Cell block, bit-line conditioning, row circuitry, and column circuitry. The SRAM Cell will store data and bit-line conditioning for stable data. We will use 3-bit addresses that approaches to SRAM, using address decoder, it makes 3-bit addresses into 8-bit data. Lastly, by using column circuitry, to reach bit-line for read the data. These functions will be manipulated by opcodes to control the data flow.

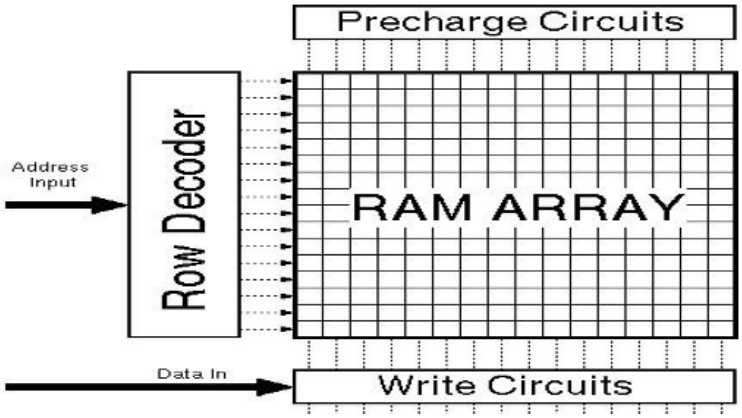

Figure 7. 8 × 8 SRAM Register File

4. Work Plan Schedule

- Logical Gates: INV, NAND, NOR, XOR – Exam 1 (Done)
- Adder: Manchester Carry Generation Carry Lookahead Adder – Exam 2 (Done)
- ALU: A+B, A-B, Increment A, Increment B, A NAND B, A NOR B, A XOR B, A NOT B (Done Schematic, Do Layout)
- 8 × 8 SRAM Register File: SRAM Cell (8 × 8 multiport 6T SRAMs), Row Circuitry (3 to 8 Decoder), Column Circuitry, Bit-line Conditioning – Exam 3
- Shifter: Logical Shifts (LSL1, LSL2, RSL1, RSL2) (Done)
- Opcodes – Control bit of the ALU, Register File, Shifter (Done ALU, Shifter)
- Latch: Store data between the data paths (Using NAND)
- Inputs: md<0:7> (memory bus data), rfs (reg. file mux selection), rfen (reg. file enable), rw (read/write), ada<0:2> (register address A), adb<0:2> (register address B), f<0:9> (ALU and shifter controls, including optional expanded function controls), clk1, clk2 (Clock Signals)
- Outputs: Y<0:7>, Cout

5. Plan

The logical function of the ALU is currently composed in a bit-wise manner using an Adder/Subtractor, which initially required a 64-to-8 multiplexer. To address this, I constructed a 3-level hierarchy of 2-to-1 multiplexers to build an 8-to-1 multiplexer. I then implemented eight instances of this 8-to-1 mux, each receiving 8 inputs and producing one output, resulting in outputs Y0 through Y7.

The 8-to-1 multiplexer assigns the correct operation result to each output based on the control signals F0, F1, and F2. While the overall floorplanning will be carried out after the SRAM is designed, I am proceeding according to Figure 1. It seems convenient to use Metal 3 to connect the control bits.

To minimize the area, I plan to pursue the most compact design possible without violating DRC rules.

6. Tentative Schedule for completing tasks

Duration	Task
3/19 – 3/26	Completing Adder
3/27 – 4/6	Completing ALU, Shifter
4/7 – 4/13	SRAM Register File
4/12 – 4/18	Completing Control Signal, Opcodes
4/19 – 5/1	Wiring, Simulation, Delay check