Austin Ebel
abe2122@columbia.edu

Full Custom 8-Bit Microprocessor

2021-05-16

1 Overview

This report will serve as a brief, relatively informal summary of the work done to design an 8-bit microprocessor as part of Professor Ken Shepard's Digital VLSI Circuits class at Columbia University. It is written after-the-fact for the purposes of my website, therefore there may be some slight errors and inconsistencies as I try to summarize what's been done around a year ago.

More concretely, this project involved the design and layout of a fully custom 8-bit microprocessor core using Cadence Virtuoso and IBM's 90nm technology. A crude overview of the design can be seen in Figure 1.

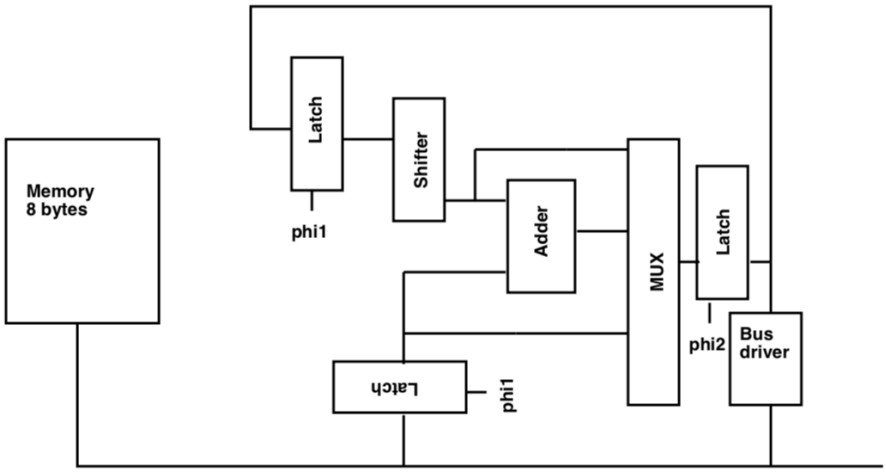

Figure 1: Datapath of the core

The design supports addition, subtraction, bit shifting, and storage/retrieval from SRAM through the opcodes and control signals shown in Figure 2. Additionally, we have clk and $\overline{clk}$ distributed throughout the core.

Opcode	Assembly	Description
000	NOP	Do nothing
001	LOAD	Mem[i] ⟵ External bus
010	STORE	External bus ⟵ Mem[i]
011	GET	Acc ⟵ Mem[i]
100	PUT	Mem[i] ⟵ Acc
101	ADD	Acc ⟵ Acc + Mem[i]
110	SUB	Acc ⟵ Acc - Mem[i]
111	SHIFT	Left logical shift of Acc

Signal	Direction	Description
$instr < 0:2 >$	input	opcode to decode
$subtract$	output	subtract control for the adder
$mux\_cntl < 0:2 >$	output	select lines for the 3-1 multiplexer
drv_enable	output	enable signal for the tristate bus driver
mem_write	output	write control for the memory
mem_read	output	read control for the memory
$shift\_bypass$	output	shifter bypass
$load\_bus$	output	load the internal bus externally
$store\_bus$	output	load the internal bus to the external bus

Figure 2: Opcodes and control signals

2 Components

Every component in this microprocessor core was designed from scratch *except* for the SRAM cell (although we still had to properly tile it & set up the read and write logic). Wires and vias were all placed by hand, and every transistor was sized for near optimal performance assuming a 2-1 resistivity ratio between PFETs and NFETs. We first designed and simulated each component within Cadence's Analog Design Environment (ADE) to verify functionality, and then moved to layout afterwards. Again, each transistor was placed by hand, and every component was passed to Calibre for

layout verification using DRC and LVS. Finally, we extracted parasitics and simulated the critical path delay for each component.

The following subsections will briefly go into the design decisions, simulations, and layout for each major component in the microprocessor core.

2.1 Addition/Subtraction

First, we designed an 8-bit two's complement ripple carry adder. To slightly reduce the delay of the critical path (carry propagation) we leveraged the inversion property,

$$\overline{S}(a, b, c_i) = S(\overline{a}, \overline{b}, \overline{c_i})$$
$$\overline{c_o}(a, b, c_i) = c_o(\overline{a}, \overline{b}, \overline{c_i})$$

to get rid of inverters in the carry logic. Also, we designed each 1-bit adder to ensure c_i was as close to c_o as possible. Finally, we used a 6-transistor XOR gate to detect overflows. A high level schematic of our adder can be seen in Figure 3.

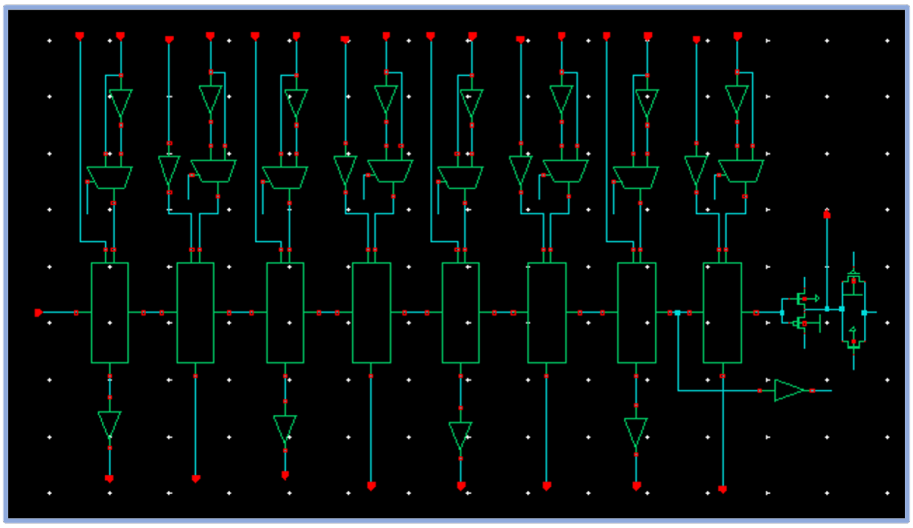

Figure 3: 8-bit adder

For each component, we began by drawing stick diagrams to visualize transistor placement. The adder was our first major design & layout, so there was an *incredible* learning curve to Cadence during the week this was due. Figure 4 shows *one* bit of our adder layout on the left, and the critical path delay (full carry propagation) of our entire 8-bit adder *after* extracting parasitics on the right. I believe we sized transistors in our adder to be relatively large because we assumed the input capacitance of our load would be large as well.

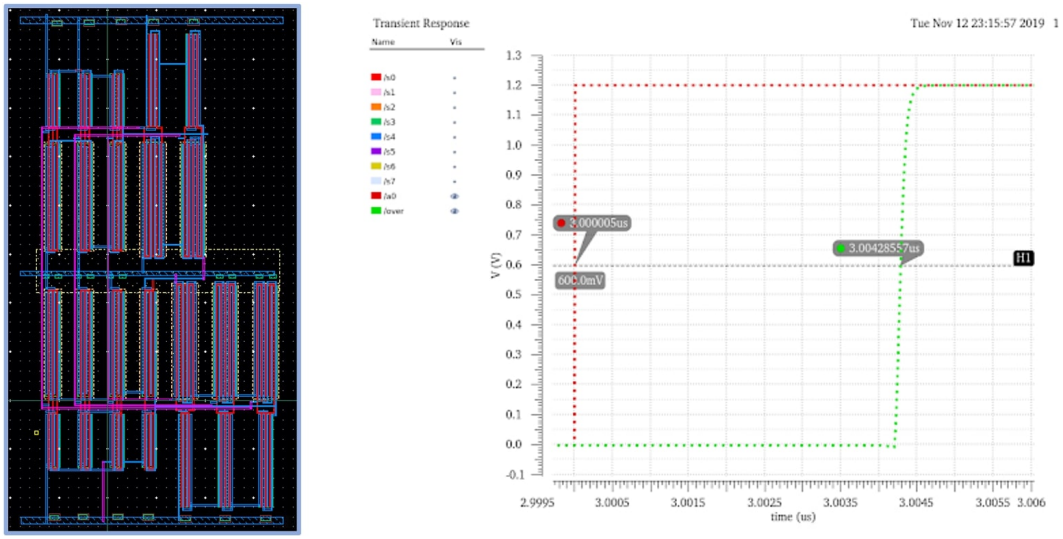

Figure 4: Single bit layout & 8-bit critical path simulation

2.2 Shifter

Next, we implemented an 8-bit logical left shifter. This was relatively straightforward and just involved three stages of multiplexers. We encoded the shift amount as a 3-bit value which served as the select line for each stage. By doing this, we could logically left shift any input value by 7 bits.

Figure 5 shows the layout of one of these multiplexers (left), and the schematic for the entire 8-bit shifter (right). Again, I believe we sized these to be relatively large due to the large load we'd be driving, but neglected to size each stage independently. A similar extraction and critical path simulation was done, resulting in Figure 6.

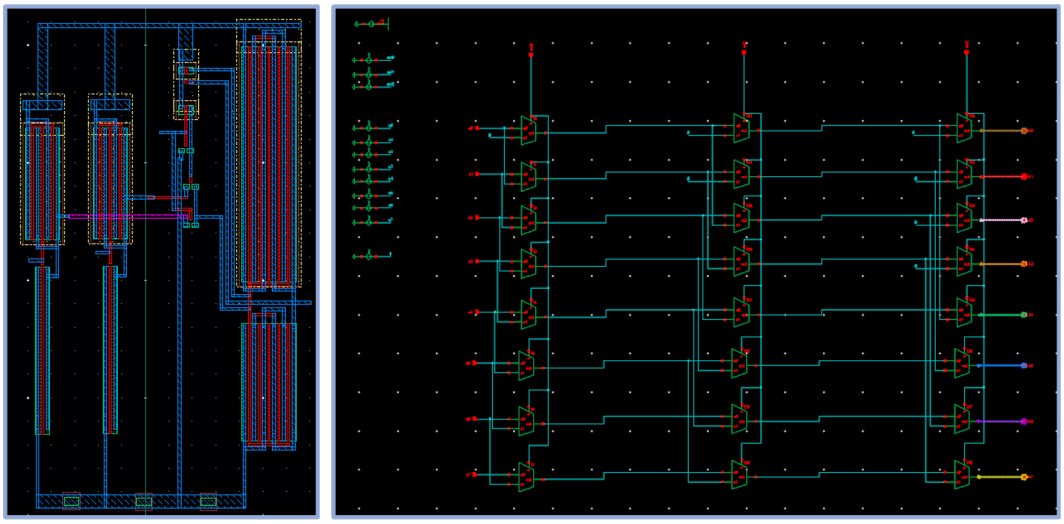

Figure 5: Multiplexer layout (left) and shifter schematic (right)

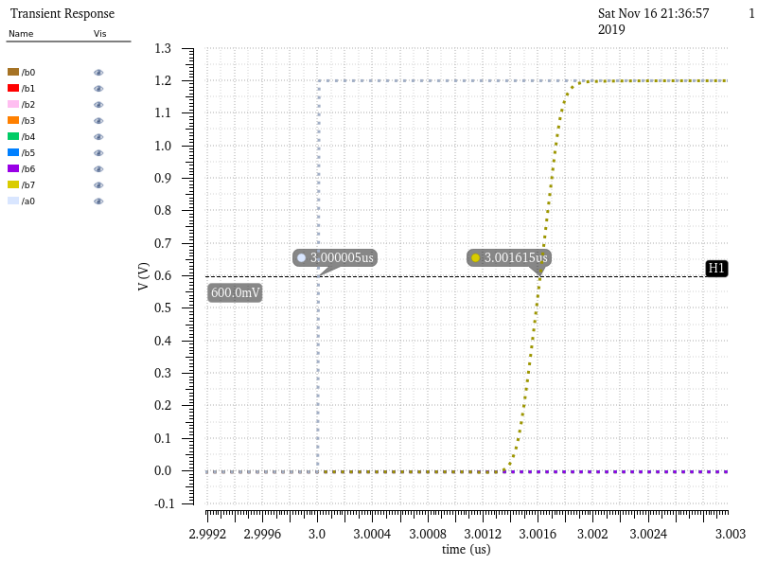

Figure 6: Shifter delay w/ parasitics

2.3 SRAM

The following week, we designed our 8-byte SRAM. As I mentioned previously, we were given a single standard SRAM cell, but had to properly tile it and design the read/write logic. *Quick aside* - I'm sure we spent well over 50 hours on this during the week it was due.

We first had to design a 3-to-8 decoder to convert the binary *address* to the appropriate wordline. Then, we tiled the SRAM cell to create an 8x8 grid and connected bit and $\overline{bit}$ lines accordingly. *AND* gates were added to the read logic for clock qualification, and a similar process was done to write data. Finally, we added a sense amplifier to ensure proper logic levels, and a tri-state buffer at the output to drive a load.

After verifying read & write functionality in Spectre, we began the layout process. Since the SRAM cells themselves were very small, significant effort went into making the read/write logic small and tileable as well. Our final schematic and layout can be seen in Figure 7. The critical path delay to read from our SRAM (including parasitics) can seen in Figure 8.

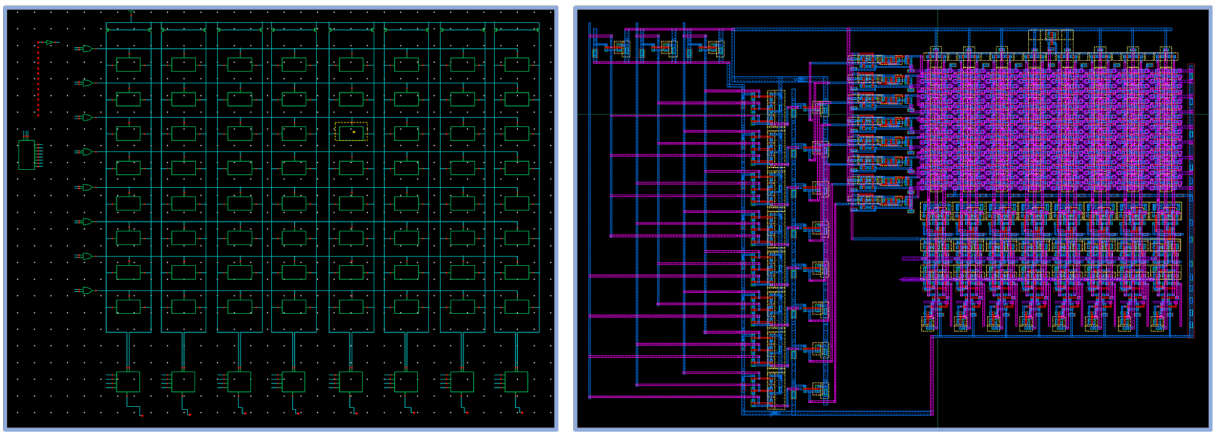

Figure 7: SRAM schematic (left) and layout (right)

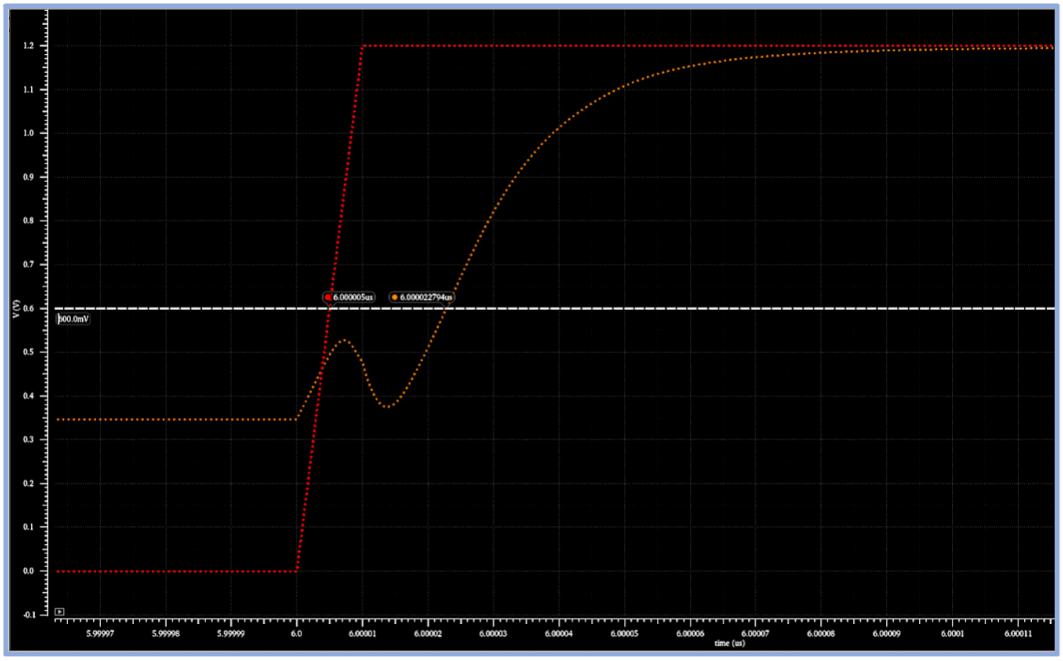

Figure 8: SRAM critical path read delay

This layout was *significantly* smaller than our adder and shifter. In fact, it was *so* much smaller that we neglected to follow our previously assumed bit pitch, and attempted to keep with the SRAM size in future layouts.

2.4 PLA

In the final week, we added a PLA to decode instructions into control signals for our datapath. Figure 9 shows our truth table for this process. We used *espresso* for logic minimization, and converted the output to a sum-of-products form for the *AND/OR* planes of our PLA. Our schematic and layout can be seen in Figure 10. I believe our design was functionally correct, however I may have slightly over-complicated some of the logic when designing it.

	i(0)	i(1)	i(2)	sub	mx(0)	mx(1)	mx(2)	de	write	read	sb	load	store
NOP	0	0	0	-	1	0	0	0	0	0	1	0	0
LOAD	0	0	1	-	1	0	0	0	1	0	1	1	0
STORE	0	1	0	-	1	0	0	0	0	1	1	0	1
GET	0	1	1	-	0	0	1	0	0	1	-	0	0
PUT	1	0	0	-	1	0	0	1	1	0	1	0	0
ADD	1	0	1	0	0	1	0	0	0	1	1	0	0
SUB	1	1	0	1	0	1	0	0	0	1	1	0	0
SHIFT	1	1	1	-	1	0	0	0	0	0	0	0	0

Figure 9: Control logic truth table

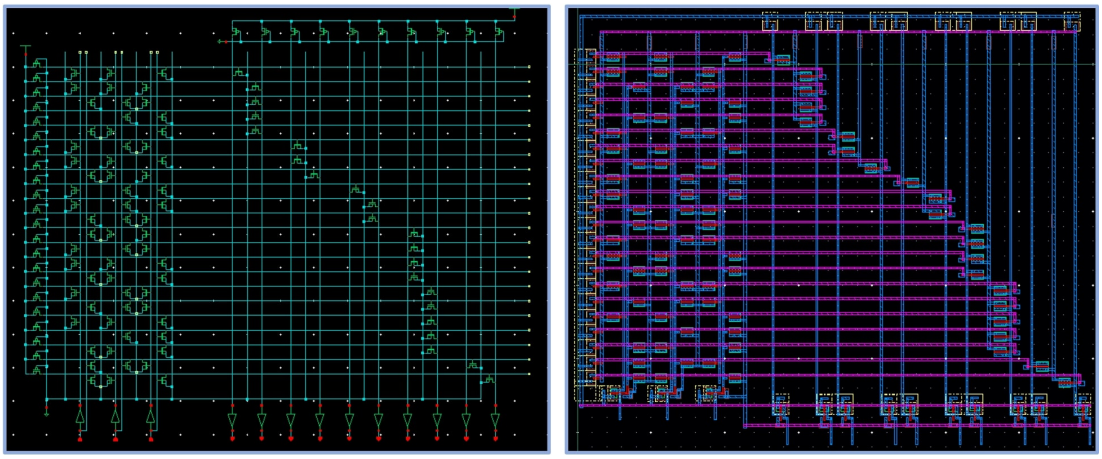

Figure 10: PLA schematic (left) and layout (right)

2.5 Other Components

In addition to the PLA, we designed an 8-bit level-sensitive latch, a 3-1 multiplexer (for 3, 8-bit signals), an 8-bit bus driver, and assembled the entire design. This was another incredibly stressful week to say the least. Layouts of all three of these can be seen in Figure 11.

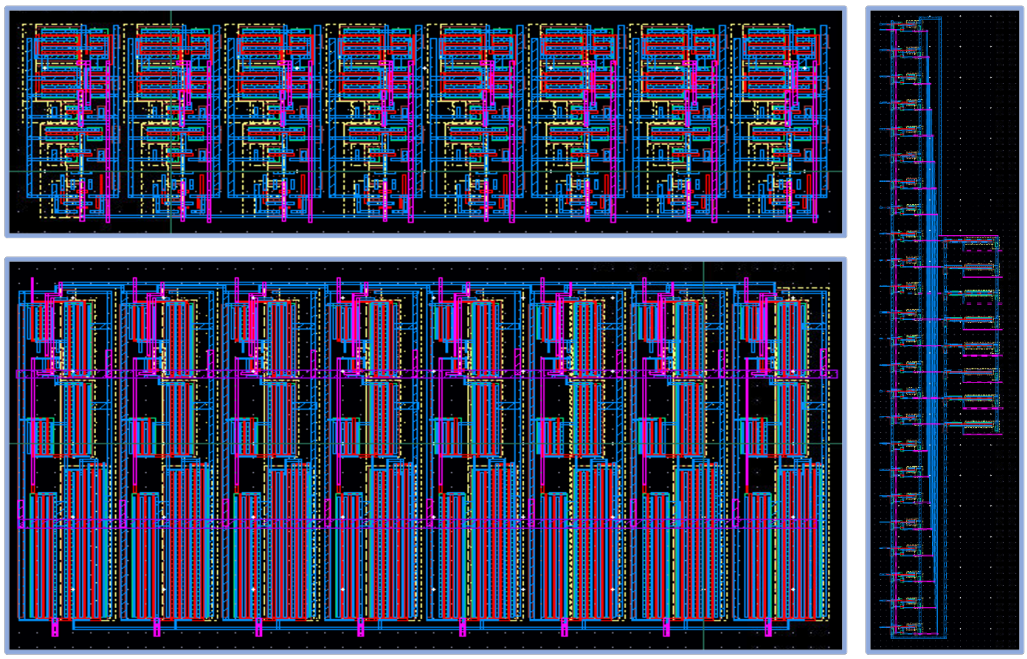

Figure 11: Latch (top left), bus driver (bottom left), 3-1 multiplexer (right)

- Latches were added before and after combinational logic blocks (separated latch design).
- The 3-1 multiplexer was used to select only one output from the shifter, adder, and memory.
- The bus driver was used as an interface between internal and external buses.

3 Final Design

We then pieced together all of the components mentioned above to form our final design and layout. There were no shortage of last minute DRC and LVS errors, but ultimately everything checked out. Finally, based on our previous delay simulations, we were able to estimate the delay of our critical path/maximum achievable clock rate to be ~ 10 ns. From this, we could also estimate the dynamic power consumption to be on the order of 10^{-5} W. The final schematic and layout can be seen in Figure 12.

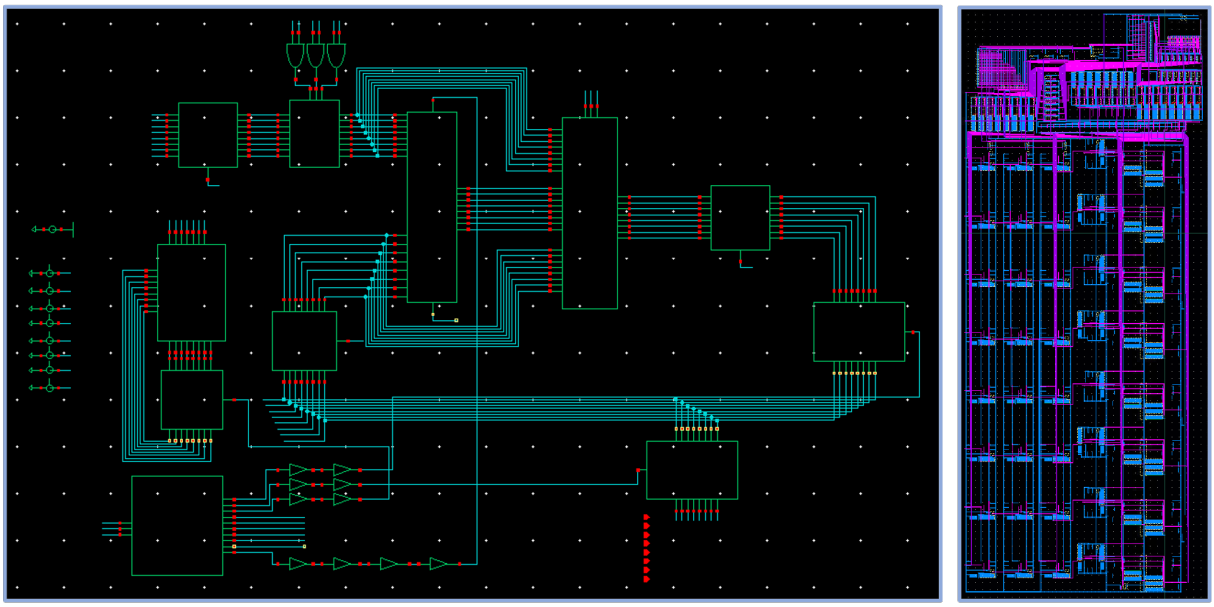

Figure 12: Final schematic (left) and layout (right)

4 Conclusions

One quick look at Figure 12, and you might be able to see that we significantly overestimated the bit pitch of our design for the adder and shifter (bottom region of the final layout). This was primarily because:

- These were our first two designs & layouts, so there was an *immense* learning curve to Cadence (especially given the time frame).
- Without a good understanding of what each future component would look like, we wanted to air on the cautious side and make the design slightly larger.
- We believed the capacitance of the adder's load would be larger than it was, so we sized components much larger than necessary.

The tiled SRAM array was the first true point of comparison, and the layouts following it were much more compact.

As I've hinted at, this was an incredibly time-consuming and nerve-wracking project. While we spent an inordinate amount of time on this in the last month of the class (roughly 40 hours/week), I would've loved to have dedicated a bit more time for further performance optimizations (and fixing past mistakes). That said, I did have to balance three other senior/grad-level classes. In the future, I'd definitely like to tackle something a bit higher level as well. Perhaps designing a processor in Verilog to get a feel for the FPGA or ASIC design flow. Ideally, something similar to this,

- https://github.com/EECS150/fpga_project_skeleton_fa20/blob/master/spec/project_spec.pdf