OctaPath - An 8-bit microprocessor data path

Michigan State University

Emmanuel ZE Butsana

Project Overview

OctaPath is the design and implementation of a complete 8-bit microprocessor data path using CMOS technology and the Cadence VLSI design suite. This project, undertaken as part of the ECE 813 design course at Michigan State University, leverages the foundational design principles developed in earlier lab exercises and expands them into a cohesive system architecture composed of a register file, arithmetic logic unit (ALU), and barrel shifter. At its core, this design realizes a functional pipeline for arithmetic, logical, and bitwise operations with fully integrated read and write access to internal memory cells. The datapath follows the reference architecture shown in Figure 1. Functional blocks communicate via control signals organized along vertically routed control lines, while data propagates throughout the datapath from block to block. Two non-overlapping clock signals, clk1 and clk2, synchronize the timing of read and write phases across the register file and ALU.

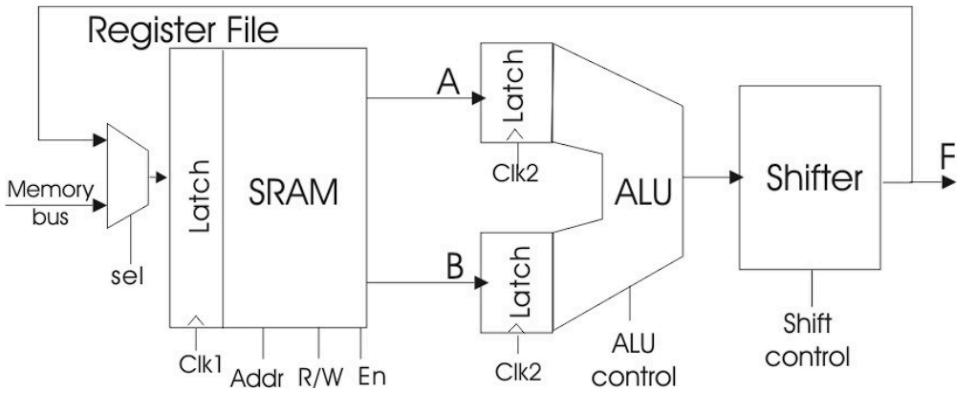

Figure 1. *Structure of microprocessor data path to be designed*

At the foundational level, the datapath is constructed from a library of primitive logic gates (INV, NAND, XOR, NOR, MUX21), each implemented with minimum-sized transistors to achieve compact layouts and reduced delay. These gates form the basis of higher-level modules such as the ALU, shifter, and memory control circuitry. The Arithmetic Logic Unit (ALU) incorporates a Manchester carry look-ahead adder as the central component, chosen for its improved propagation delay characteristics over ripple carry structures. This unit supports a broad range of operations, including logic functions (NOR, XOR, NAND, NOT) and arithmetic functions (increment, decrement, add, subtract). These functions are controlled via encoded opcodes using a 3-bit control scheme, with a careful encoding to minimize logic complexity. Complementing the ALU is a logarithmic barrel

shifter, enabling data manipulation operations such as logical shifts and rotates in both directions. While not strictly required by the project specification, the inclusion of rotate operations provides enhanced bit manipulation capabilities that could prove valuable in numerous scenarios. The register file is implemented as an 8×8 SRAM array utilizing 6-transistor (6T) cells. It features two simultaneous 8-bit read ports and one 8-bit write port, allowing dual-operand ALU operations within a single read cycle. The write path supports input either from the ALU or an external memory bus via a multiplexer and an input latch. Clocked synchronization and enable gating ensure proper timing between operations, with high-impedance outputs enforced during write cycles to prevent bus contention.

In summary, OctaPath represents a comprehensive and custom-built datapath solution, distinguished by its modularity, timing-aware layout, and extended functional capability. The design effort reflects a holistic integration of digital logic, VLSI layout strategies, and system-level architecture, culminating in a datapath that meets the requirements for an 8-bit CMOS processor.

Design Methodology

The development of OctaPath followed a structured, bottom-up design methodology rooted in hierarchical abstraction and progressive integration. The approach focused on achieving correct functionality at each design stage while continuously optimizing for layout area, logical delay, and routing compatibility. Early-stage planning was critical to defining a scalable bit-pitch, signal routing strategy, and modular design flow, all of which underpinned the project's successful implementation. At the outset, a complete library of primitive CMOS logic gates was designed, including inverters, 2-input NAND, NOR, XOR, and multiplexers. Each gate was constructed using minimum-sized transistors, prioritizing compact area while maintaining acceptable drive strength and noise margins. The layouts for these gates adhered to a common height and consistent pin placement to ensure reusability and vertical pitch matching across higher-order modules. DRC and LVS verification was performed at this level before their instantiation into more complex cells.

The first major composite module designed was the Manchester carry look-ahead adder, the central computational component of the ALU. The adder was constructed by vertically stacking two 4-bit Manchester carry look-ahead adders, with each 4-bit cell comprising its sum, propagate/generate, and carry computation circuits. This architectural arrangement allowed for improved visualization of carry propagation and facilitated layout

debugging during simulation and physical verification. Care was taken to route internal signals efficiently. All carry control logic was implemented hierarchically and validated through functional simulation before layout. Following the adder, the ALU control logic was designed to support encoded selection signals that correspond to a well-defined opcode truth table. Logical and arithmetic operations were integrated via multiplexing at the ALU output, controlled by the decoded function inputs.

The barrel shifter was implemented using a logarithmic architecture, enabling single- and multi-bit shifts and rotations in both directions. The design employed a cascade of 2:1 multiplexers organized in a three-stage shift network, each stage conditioned on one bit of the shift amount. The register file was implemented as a dual-read, single-write 8×8 SRAM array. The core memory cell utilized a 6T configuration, chosen for its density. Peripheral circuits, including address decoders and column circuitry, were developed independently and integrated into a single memory block. A multiplexer at the write port allowed data selection from either the ALU or the external memory bus, while input and output latches ensured correct timing across clock domains. The enable signal was carefully integrated into both address decoding and output gating logic to ensure correct behavior during read and write cycles, with high-impedance states enforced during non-enabled periods.

Throughout the design process, hierarchical schematics and modular layout discipline were maintained. Each subcomponent was functionally simulated in isolation and verified post-layout against the extracted netlist. Design rule compliance (DRC) and layout-versus-schematic (LVS) checks were conducted iteratively at each level to ensure correctness and consistency. The integration phase brought together all verified modules into the full 8-bit datapath. Final checks included full functional simulation of the complete datapath and characterization of delay for the slowest logic paths. Thus, the design methodology prioritized correctness, modularity, and optimization at every stage, reflecting a deliberate and methodical approach to complex digital system design in VLSI.

Design and Results

The foundation of the datapath was established through the construction of a library of primitive CMOS logic gates, together with a few other basic components. These designs focused on layout efficiency and reusability in higher-level circuitry. At this stage, designs followed a schematic-to-layout flow outlined earlier in the course. The first gates to be developed were a 1-bit inverter gate, a 2-input NAND gate, a 2-input XOR gate, a 2-input NOR gate, and a 2:1 multiplexer. Of these, the most challenging and best example of the

design procedure followed was the XOR gate. To simplify the design, care was taken to ensure the gate was implemented without splitting the active regions for either the pMOS or nMOS transistors, as shown in Figure 2. This cell maintained a 21 µm pitch, a standard held across all logic gates, and a width of 16.8 µm. On top of these basic logic gates, a few more compound logic gates were developed to make later development easier. Firstly, non-inverted 2-input logic gates (AND2 and OR2) were developed for use in both the masking circuitry of the barrel shifter and the carry generation circuits of the Manchester adder. On top of these, 8-bit logic gates, together with an 8-bit 2:1 multiplexer, were developed for use in the ALU. These were made by chaining together several of their 1-bit counterparts in parallel.

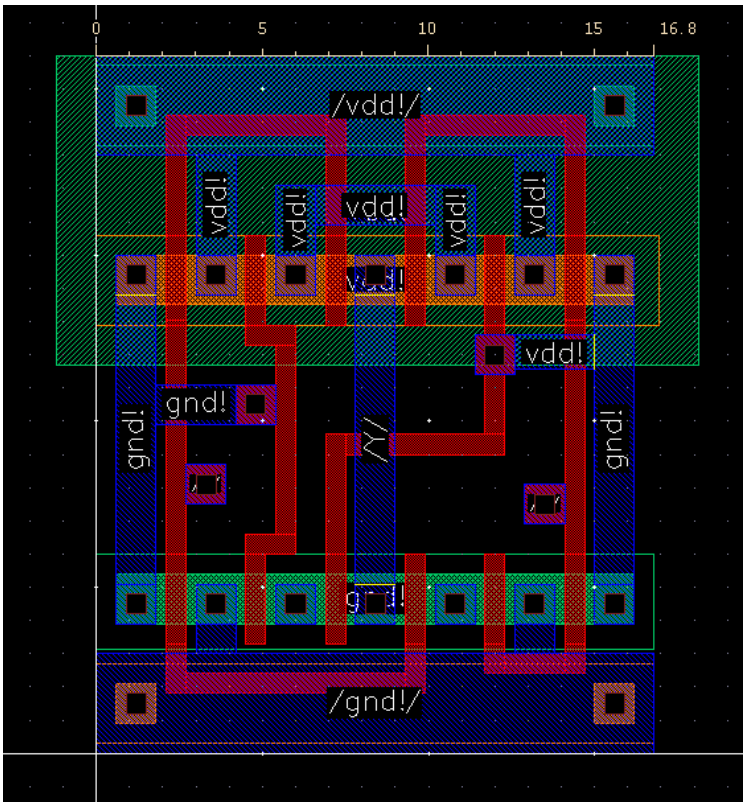

Figure 2. *Layout of a 2-input XOR gate*

The first component to be built using these primitive gates was the 8-bit Manchester Carry Generation Carry Lookahead adder, which was made by chaining together two smaller 4-bit Manchester Carry Generation Carry Lookahead adders, each of which implemented its own Manchester adder cell and Carry Lookahead cell. Looking at the Carry Lookahead cells, these work by generating a group *generate* and *propagate* signal for given 4-bit blocks and using these to generate a *cout* signal. While the *generate* and *propagate* signals could instead be taken from the Manchester adder cell instead of being regenerated here, the chosen

approach allows modular testing and isolated debugging of the carry logic without dependency on the correctness of the arithmetic unit, thereby simplifying verification and layout reuse. Similarly, the Manchester adder is made by chaining together 4 1-bit Manchester adders, inverting the *cin* bit into the first one as well as the *cout* bit out of the last one. With this, we end up with the layout shown in Figure 3 for our 8-bit Manchester Carry Generation Lookahead adder.

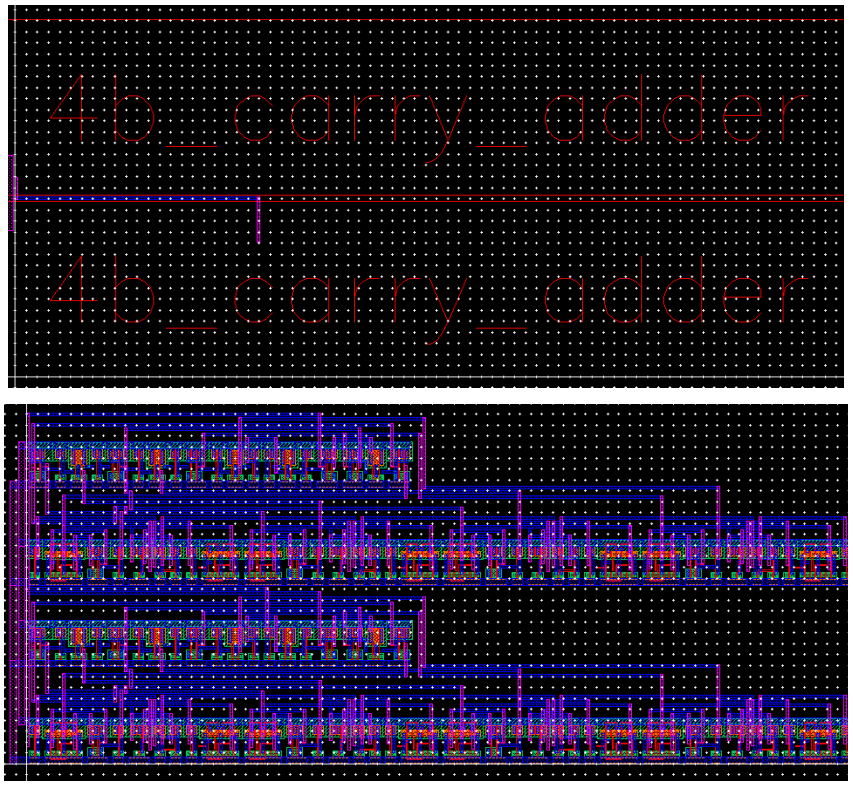

Figure 3. *8-bit Manchester Carry Generation Lookahead adder (hierarchical and full view)*

The adder, together with careful use of op-codes and some circuitry to process the input *B*, formed the backbone of the ALU arithmetic unit, which implemented some of the operations shown in Table 1. The S_1 and S_2 bits were used to select what value of B would be chosen based on the operation being performed, which can also be seen in Table 1. S_2 was also used to provide the *cin* bit of the adder. The circuit to obtain the appropriate B output simply consisted of an 8-bit 4:1 multiplexer, which took as input all the values of B shown below. To implement the logic operations of the ALU, an 8-bit 4:1 multiplexer was used to select the appropriate output from the 4 8-bit logic gates. The S_0 bit then served as a way for the selection of logic or arithmetic output. Using this selection logic, the ALU could be implemented using a single adder for the arithmetic unit, together with a single logic unit. For testing, all ALU operations were simulated with inputs A = 1111 1000 and B = 0001 1111.

With these, the slowest logic operation was found to be the XOR function while the slowest arithmetic operation was found to be the subtraction operation.

S_0	S_1	S_2	B value	Operation
0	**0**	**0**	B	A + B
0	**1**	**0**	0000 0001	INCREMENT A
0	**0**	**1**	1111 1110	DECREMENT A
0	**1**	**1**	~B	A - B
1	**0**	**0**	N/A	INV
1	**0**	**1**	B	A XOR B
1	**1**	**0**	B	A NAND B
1	**1**	**1**	B	A NOR B

Table 1. *ALU operations and their opcode*

The next component to be made for the datapath was the SRAM. Based on the design requirements, the SRAM was to have two read ports and one write port. Thus, the 6T cell design as selected, as it could be used with two read ports, by reading from both *bit* and $\overline{bit}$, and one write port, by writing on both *bit* and $\overline{bit}$. Furthermore, this cell provided greater density than the alternative 8T cell. To support stable read and write operations, a bitline conditioning circuit was designed that utilized the clock signal to precharge the bitlines and control the timing of wordline activation. Instead of reusing existing transmission gate cells for the column circuitry, the design was redone at the transistor level to ensure greater density. This allowed us to obtain a column circuitry cell with dimensions 26.4 μm by 36.4 μm. The same mantra was followed for the address decoder layout, allowing us to obtain a cell with dimensions 65.4 μm by 50.7 μm. Integrating all these components into a complete SRAM unit, transmission gates were used to determine what values should be written to the address bits of port B. When the *rw* signal is 0, indicating a read operation, transmission gates connect the address bits of port B to those being written by a user. When the *rw* signal is instead 1, indicating a write operation, the address bits of port B are connected to those of port A because, as previously mentioned, the 6T cell requires us to write on both bit and $\overline{bit}$ to write a value to the cell. Shown in Figure 4 is the completed schematic of the entire

SRAM. The performance of the SRAM unit was tested over two full clock cycles, writing 0000 0000 to address 000 and reading it from port A during the first cycle, and writing 1111 1111 to address 111 on the second cycle and reading it from port A, while port B reads the value at 000. With this simulation, we find the propagation on port B to be approximately 40.5178 ns while the propagation on port A is approximately 40.4487 ns.

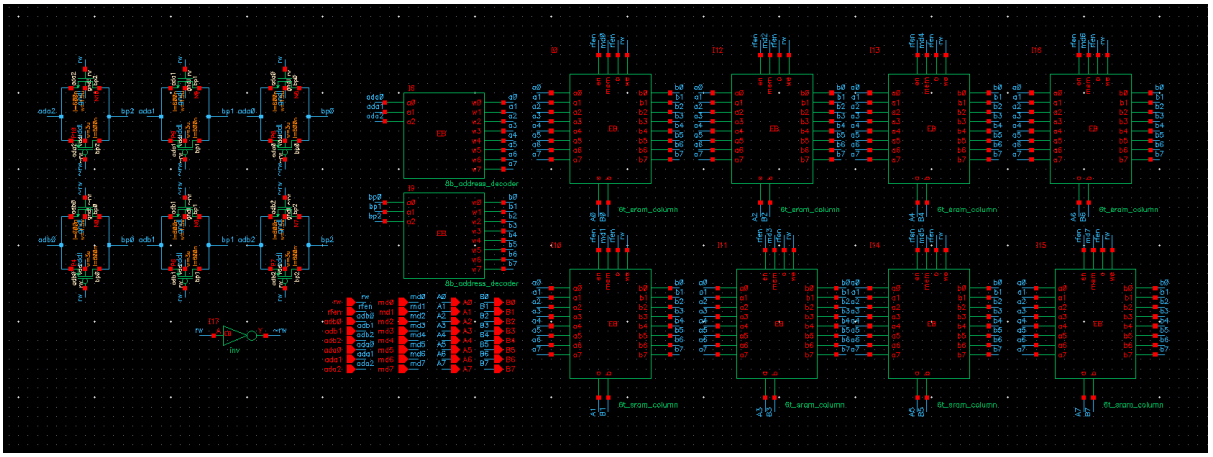

Figure 4. *8x8 SRAM Schematic*

The last component of the SRAM to be tested was the logarithmic barrel shifter. To allow for both shift and rotate operations, the shifter was made using 3 key components: a shifter that rotates 8-bit inputs by 1 through 7 left or right, a decoder to give the mask bits for shifting, and some circuitry to apply the mask to the rotated bits in the case of a logical shift operation. Let us look at these constituent parts individually. Ignoring the data it takes in, the shifter takes as input 3 bits (k_0, k_1, k_2) to determine the extent of the rotation and a single bit to determine whether it is a rotation to the left or the right. It consists of a preshift level followed by three rotation stages, with each stage's shift being determined by applying the exclusive OR operation to the level's input *k* and the left shift bit *left*. The decoder takes as input the 3 shifting bits (k_0, k_1, k_2) and decodes this thermometer code to produce the appropriate masking bits. The masking circuitry takes these masking bits and applies them to our shift operations if they are not a rotation. The mask application circuitry for a singular bit is shown in Figure 5. As expected, it takes as input *shift* (to determine wether this is a rotate or shift operation), *left* (to determine whether this is a shift left or right), *right* (the right mask), *left* (the left mask), and *rot* (the bit obtained from a rotation). The performance of the shifter was tested by providing a data input of 1111 0000 and rotating it right by 2, and then changing the input to 0000 1111. With this simulation, we find the propagation to be approximately 25.653 ns.

The final design could not be completed on time due to unforeseen circumstances. A list of all the cells created as part of this design, however, with appropriate timing analysis included, is shown in Table 2. Furthermore, the complete layout for the shifter, together with partial layout completed for

the ALU, are shown in Figures 4, 5, and 6. The slowest propagation delay of the data path (through the ALU and the shifter), found through functional simulation, was approximately 36.526 ns. The slowest logic function was the XOR function, with a propagation delay of approximately 15 ns. The slowest arithmetic function was the subtract operation, with a propagation delay of 21.981 ns. This delay is obtained through functional simulation, as a layout putting together the ALU and shifter was not completed. We cannot provide a complete physical area for the data path layout or the total number of transistors required, as individual components were not put together to form a complete data path

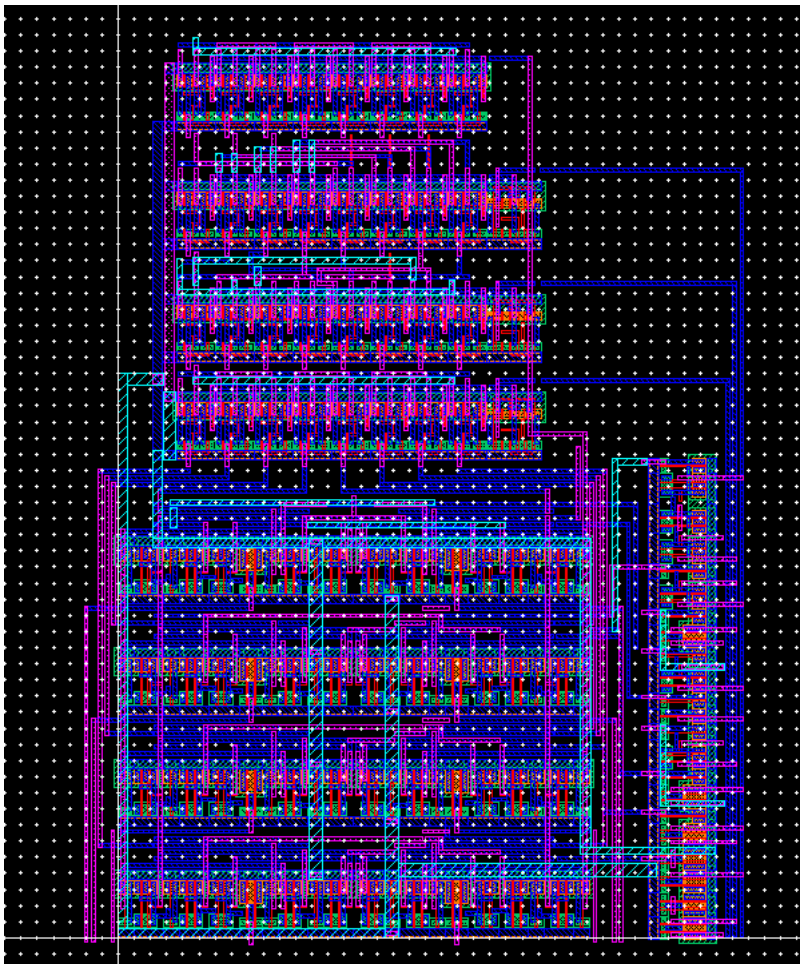

Figure 4. *A complete shifter layout*

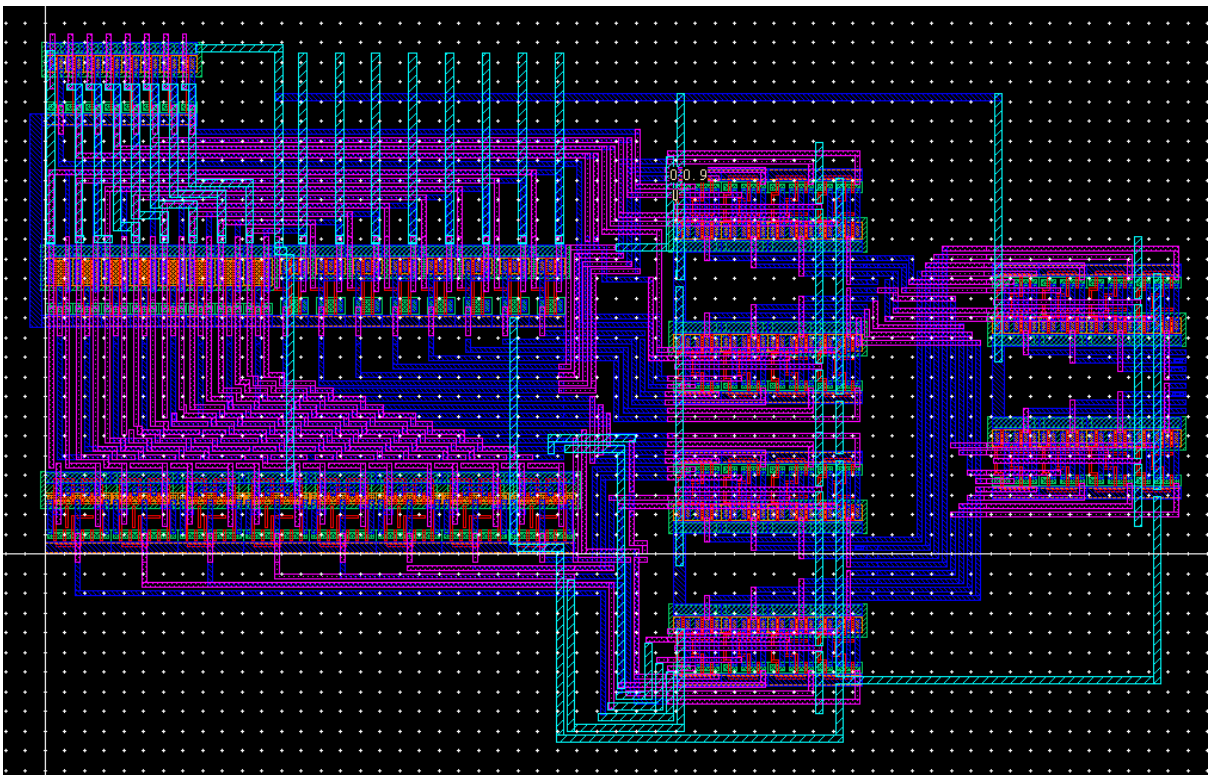

Figure 5. *Logic unit of the ALU*

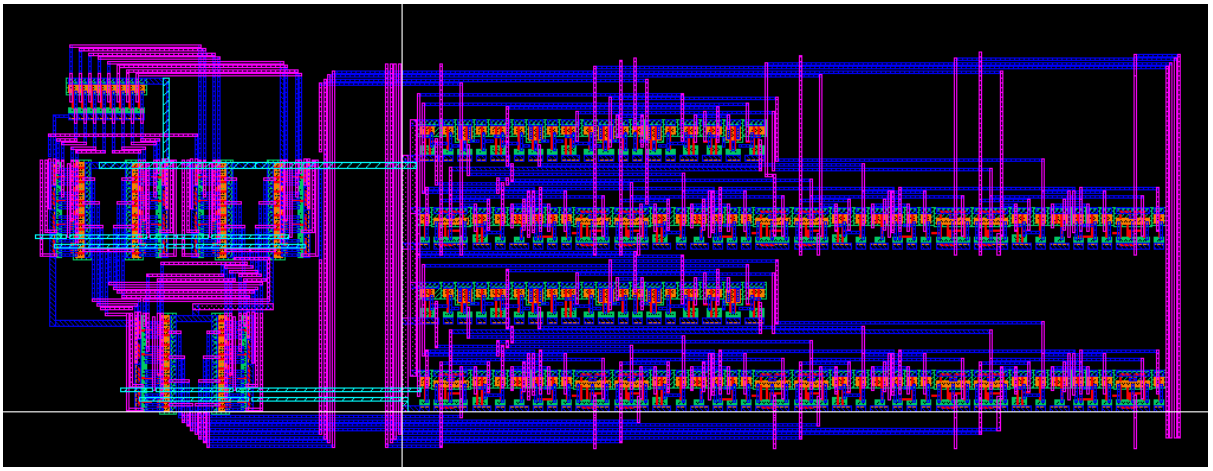

Figure 6. *Arithmetic unit of the ALU*

Conclusions

The design of OctaPath, a complete 8-bit microprocessor data path, has offered a comprehensive and instructive exploration of hierarchical VLSI system design using CMOS technology. By incrementally developing and integrating key subsystems, including the arithmetic logic unit (ALU), barrel shifter, and dual-read, single-write SRAM register file, the project demonstrated how lower-level logic primitives can be composed into higher-level architectural constructs that support a wide range of computational tasks. The final design

meets all specified functional requirements, including arithmetic and logical operations, multi-bit shifting and rotation, and synchronized memory access. The design methodology employed emphasized careful planning, modular construction, and rigorous post-layout verification. Layout optimization, pin alignment, and cell reuse were central to maintaining consistency and achieving integration across components. The inclusion of bidirectional rotation in the shifter, while beyond the minimum project requirements, added functional richness and illustrated the system's extensibility.

Nevertheless, the project was not without its limitations. Due to external scheduling conflicts with other coursework, final layout integration of the complete datapath and the ALU could not be completed. Despite this, all foundational and composite cells were successfully implemented, verified through simulation, and passed DRC and LVS checks where applicable. Timing analysis was performed for critical paths, and extracted delays remained within acceptable bounds as defined by the project specifications.