4-Bit RISC Processor Design and Physical Layout Using Cadence Virtuoso

Final Technical Design Report

Department of Electrical and Computer Engineering

Michigan Technological University

Fall 2025

Design Team:

Cecil Takunda Mhike

Jonathan Joseph

Vishrut Chokshi

Table of Contents

1. Abstract
2. Introduction
3. Project Objectives
4. Design
5. Results and Discussion
6. Conclusion
7. References

Abstract or Summary

This report and documentation describe all the steps taken and procedures to produce the architectural specifications, development of a functional schematic, the physical layout and the integration to the system level of a 4-Bit RISC processor built using the Cadence Virtuoso software utilizing the skywater130 PDK. A team compromising of 3 people namely Jonathan Joseph, Cecil Takunda Mhike and Vishrut Chokshi designed and completed the whole CMOS digital pipeline, which includes the conceptual architecture to the resultant functional schematics, individual block layouts, Pegasus DRC, ERC, LVS verification and preparation for the whole full chip floor planning. The primary design objectives were accomplished which include the development of a working Datapath, validation of individual functional blocks, manufacturable layouts and cohesive integration strategy for final verification. The 4-Bit RISC processor design project displays rigorous engineering practice, which is very much compatible with the VLSI design requirements.

Introduction

The resultant goal of the 4-Bit RIC processor project was to design and build functional blocks utilizing industrially recognized VLSI standards and processes while recognizing a realistic digital network. Teamwork aided to the development of the RISC processor from the ground up, implementing the required subsystems which are the Arithmetic Logic Unit, Register File, Barrel Shifter, Control Logic and the Multiplexer Network while ensuring that these individual blocks satisfied the electrical, logical and physical design specifications.

To meet the criteria for this RISC processor design, project topologies relating to circuitry that ensured simplified understanding of the behavior of the very large scale integration systems were utilized. All this was done utilizing the Skywater130 CMOS rules, the Pegasus bases DRC/VLS/ERC verification and for full chip floor planning and routing validating layout manufacturability and schematic matching. At the top level, the strategies relating to the floorplan and routing were put together forming a complete chip therefore creating an integration ready layout of the 4-Bit RISC processor.

Utilization of tests that are focused on operations done by the Arithmetic Logic Unit, the shifts by the Barrel Shifter, Control Paths and transfers by the Register was done. The Arithmetic Logic Unit showcased patterns involving logical and arithmetic functions that are simplified. One the Datapath the walking zero and walking one patterns were some of those test patterns that were tested for.

Sequences generated by the instructions that stress the control logic and the data flow were tested and all the tests combined and joined gave validation and assurance towards the intended processor operation being achievable.

Overall, the entire lifecycle development of digital integrated Circuits is shown in this project from the specifications of the architecture to the schematic entry and the simulation of the layout. From Week 11 to week 14 the required goals were utilized to organize efforts among our 3-person team and assured that progress was made toward the development of a validated and integration ready processor design.

Project Objectives

The primary objectives of this project were to:

- Define a clear 4-bit RISC architecture, encompassing Datapath, control, and memory interactions.
- Design all schematics ensuring that they have functional behavior and reliable simulation performance.
- Create and develop manufacturable layouts following SkyWater130 technology principles.
- Verify all blocks using Pegasus DRC, ERC, and LVS methodologies.
- Plan for full-chip integration, including pin alignment, routing strategies, and floorplanning principles.
- Document the entire design process using professional, graduate-level engineering standards.

DESIGN (Schematic Section)

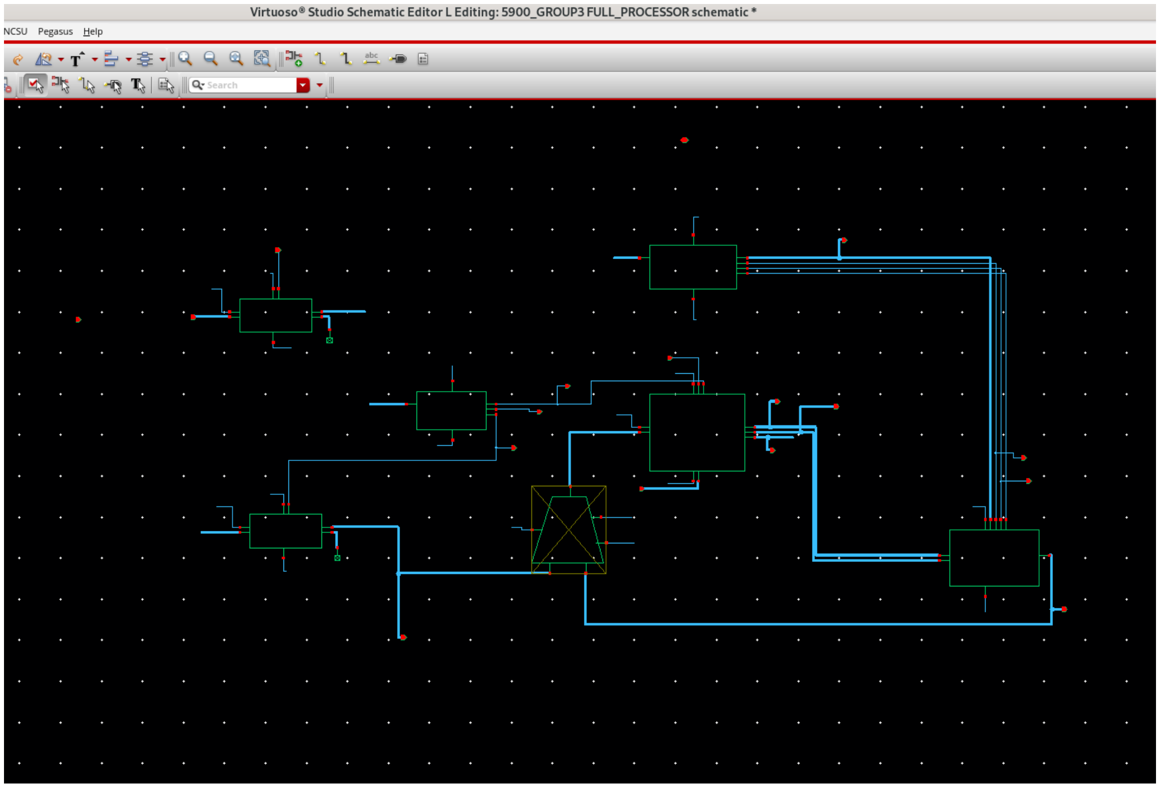

Virtuoso® Studio Schematic Editor L Editing: 5900_GROUP3 FULL_PROCESSOR schematic *
NCSU
Pegasus
Help
Search

Figure above shows the Full 4-Bit RISC processor schematic designed by the team

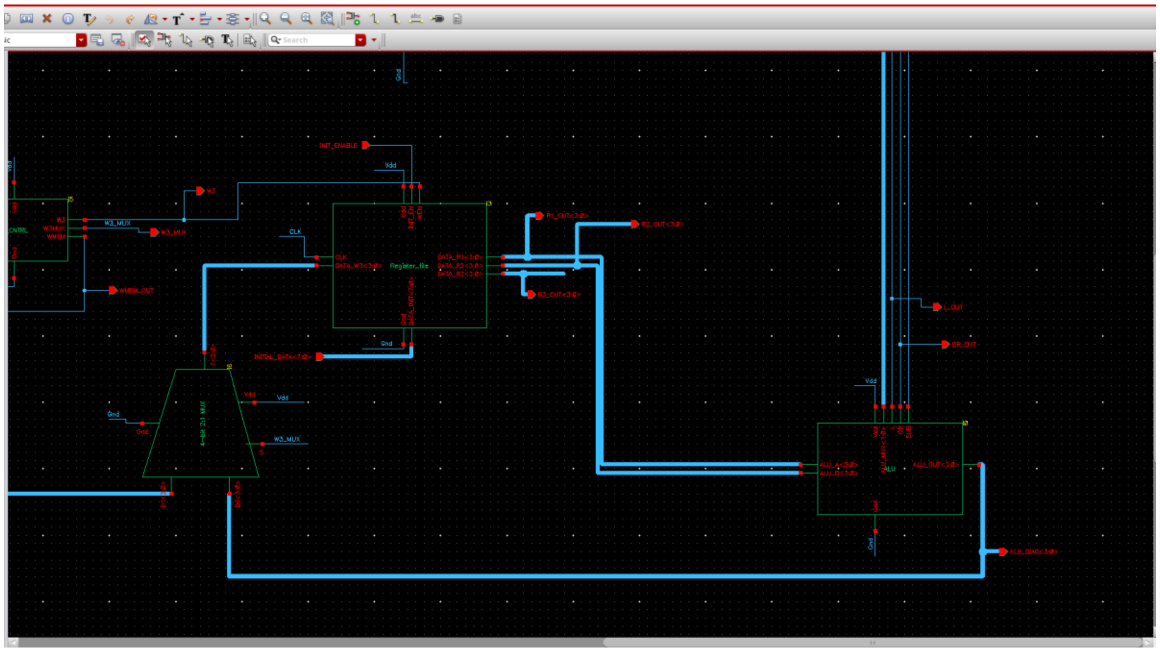


Figure above shows the close-up view of the 4-Bit RISC processor Schematic

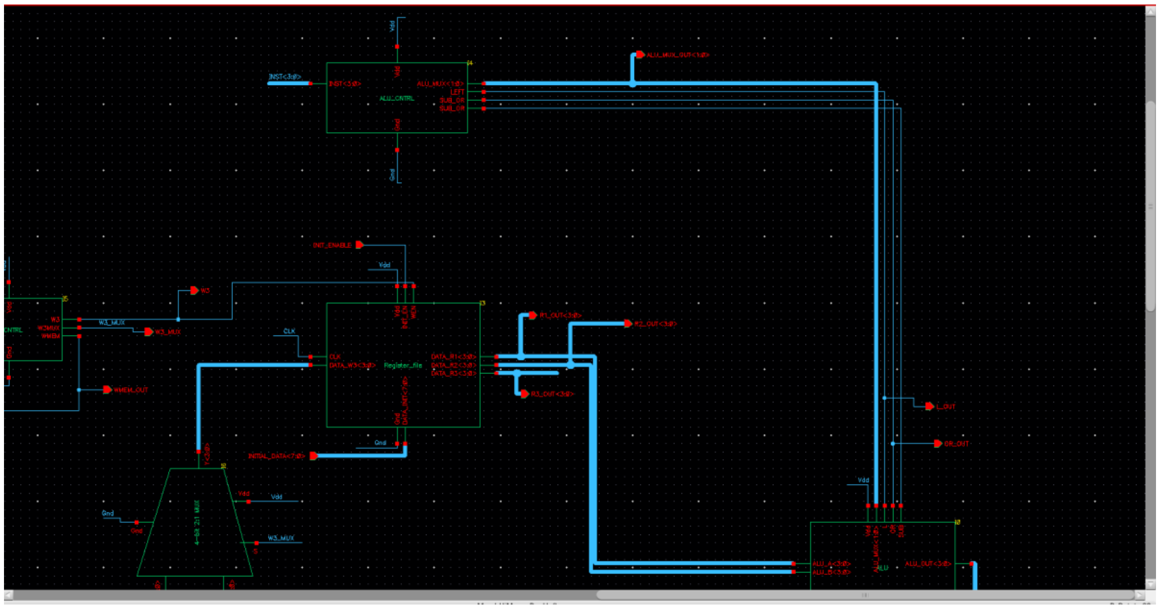
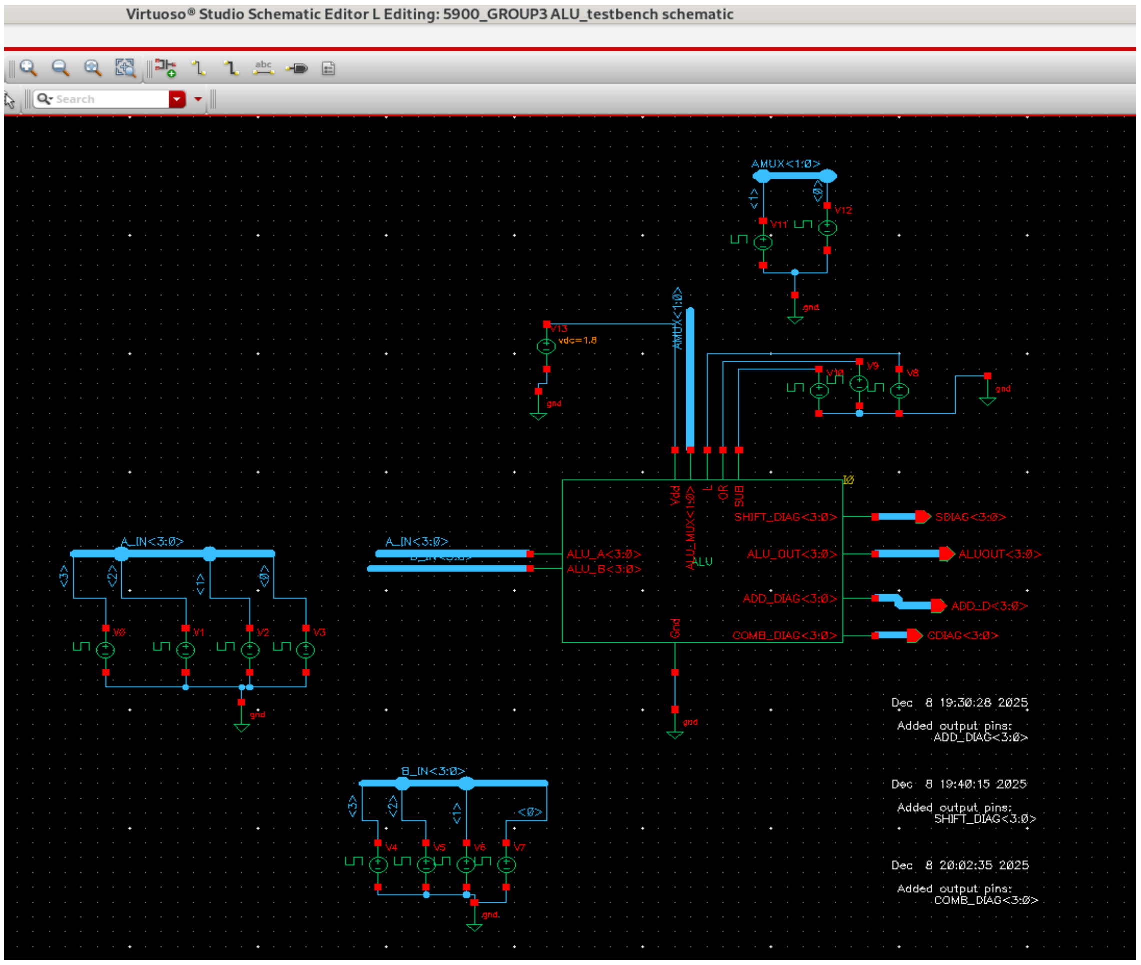


Figure above shows the Full 4-Bit Processor Schematic close up

The instruction path, control logic, register file, multiplexers, and ALU are all combined into just one synchronous Datapath in the accompanying full 4-bit RISC processor layout. The 4-bit instruction nibble INPUT_INST<3:0> is initially recorded in an instruction register towards the left, resulting in INST<3:0>. The RW_CNTRL block decodes this instruction word and produces writing back enables, register reading/writing selects, and ALU operation controls that power the remainder of the system.

The instruction path, control logic, register file, multiplexers, and ALU are all combined into just one synchronous Datapath in the accompanying full 4-bit RISC processor layout. The 4-bit instruction nibble INPUT_INST<3:0> is initially recorded in an instruction register towards the left, resulting in INST<3:0>. The RW_CNTRL block decodes this instruction word and produces writing back enables, register reading/writing selects, and ALU operation controls that power the remainder of the system.

The hierarchical ALU block on the right generates ALU_OUT<3:0> along with other function-specific outputs (such as OR_OUT and L_OUT) after receiving its 4-bit operands from the chosen register outputs. The global CLK synchronizes all data movement, and the decoded enable signals (INT_ENABLE, register enables) gate write operations such that state modifications only take place on the designated clock edges. By leveraging the previously established register, mux, and ALU cells, this integration technique maintains the design's structured hierarchy while producing a clear, bus-based top-level Datapath that is simple to incorporate into layout and validate in the SkyWater130 PDK.



Full Processor Testbench used for simulation results

The above figure shows the schematic testbench used with Maestro simulation to test the functionality of the processor. Inputs for R1= 0010 R2= 0001 were chosen to show the effect of a shift. The simulation clock operated at a period of 100ns and correctly executed all of the instructions. There are two signals used for the first two clock cycles, to enable R1 and R2 so that they can be initialized, and an instr_en signal that is off initially that deactivates the instruction input register.

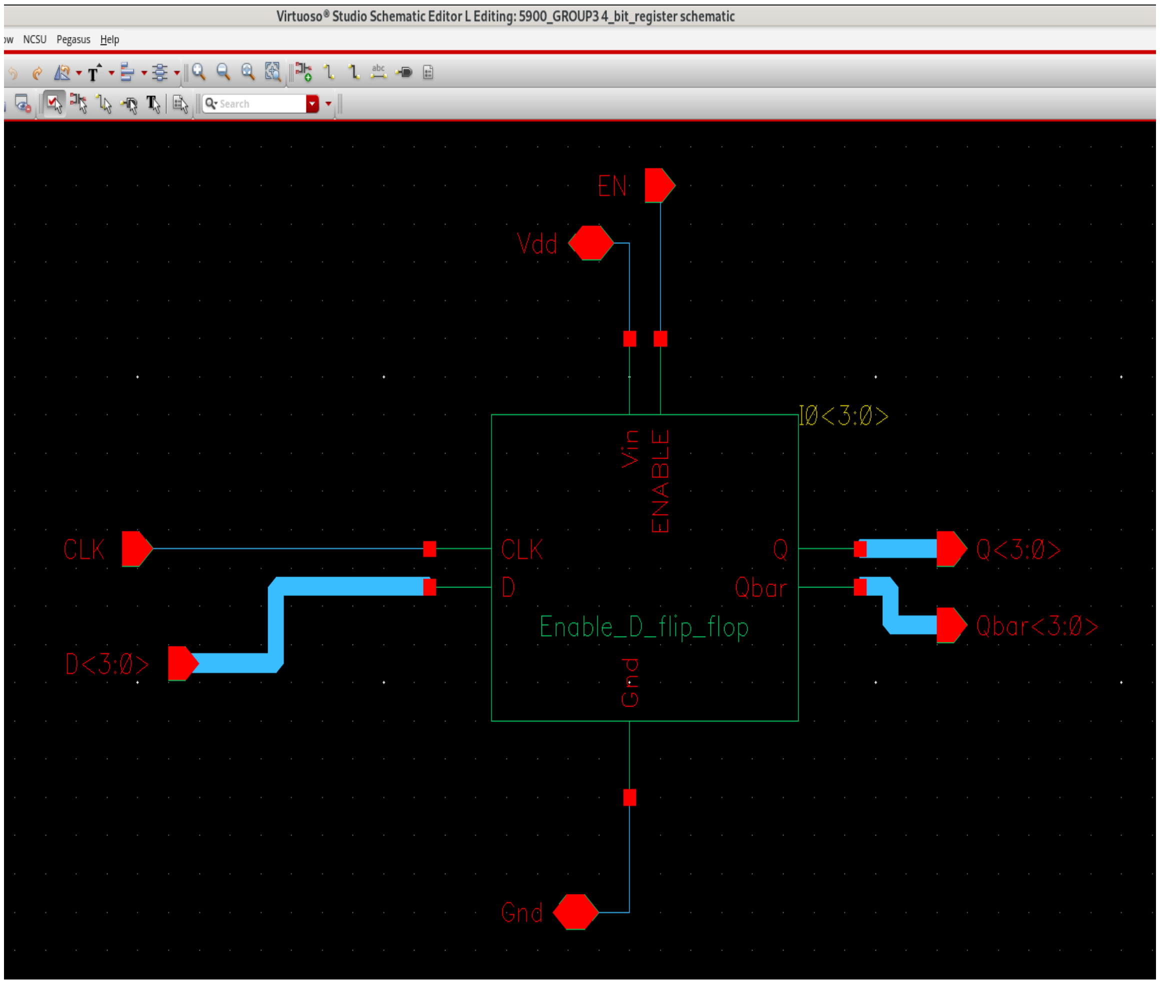


Figure above shows the 4Bit Register Schematic designed by the Team

The 4-bit registers that make up the processor's storage components are constructed from an enabled D flip-flop macro (Figure Y). The schematic creates an individual hierarchy cell called Enable_D_flip_flop and reveals data, a timer, and controller as buses. Thus, since the EN signaling is asserted, the input bus D<3:0> is sampled on the active clock edge, then the resultant bus Q<3:0> displays the outcome of the value.

Additionally, Qbar<3:0> opposite outputs are offered for usage in down logic when reversed signals are beneficial. The cell is simple to duplicate for several registers in the file thanks to a shared CLK input and common Vdd/Gnd rails. An edge-triggered, enable-controlled topology prevents level-controlled transparency problems, guarantees synchronous writings under FSM control, and fits neatly into the SkyWater130 CMOS layout as a small, bused register block.

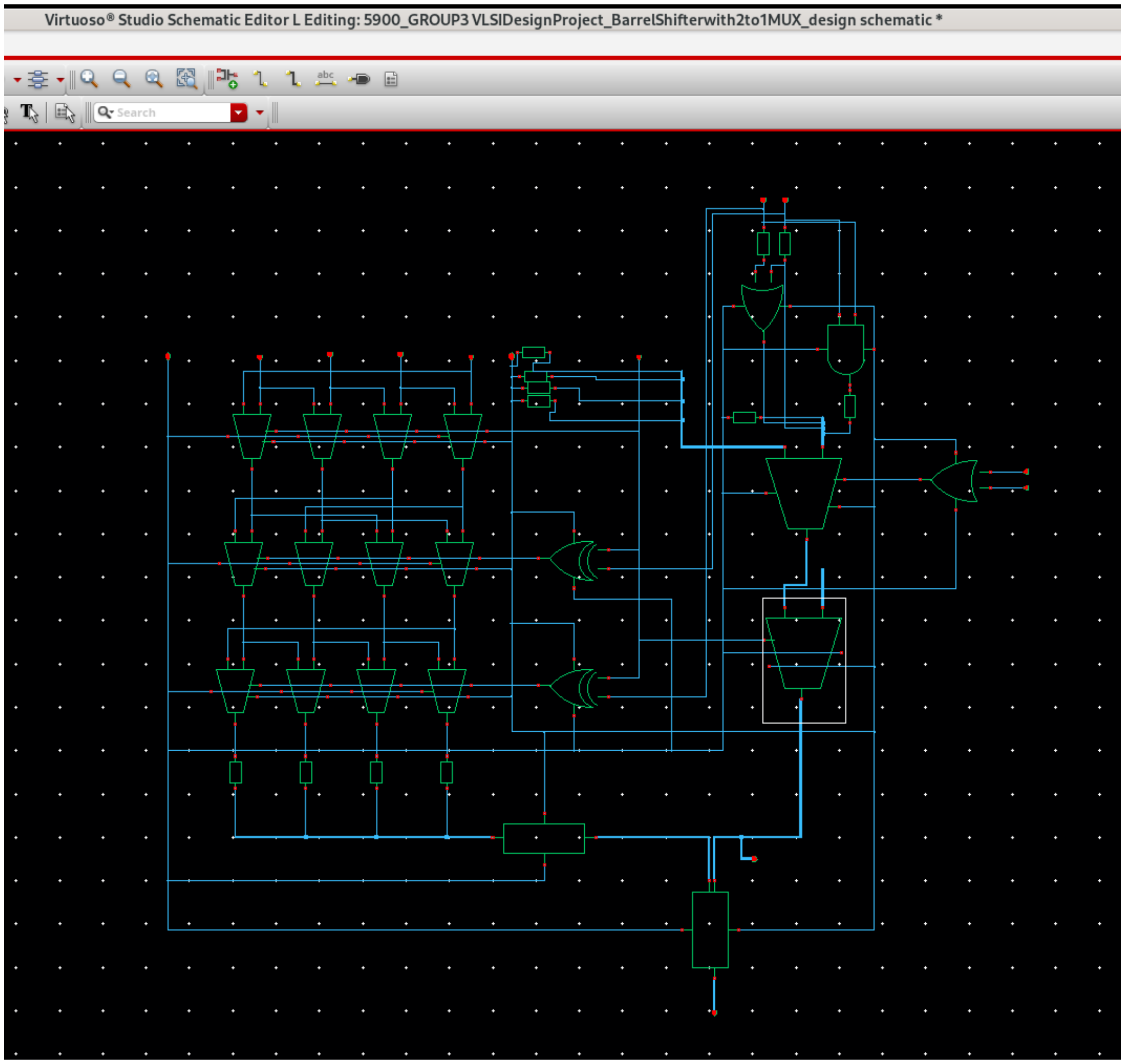

Virtuoso® Studio Schematic Editor L Editing: 5900_GROUP3 VLSIDesignProject_BarrelShifterwith2to1MUX_design schematic *
Search

Figure above shows the Barrel Shifter schematic designed by the team.

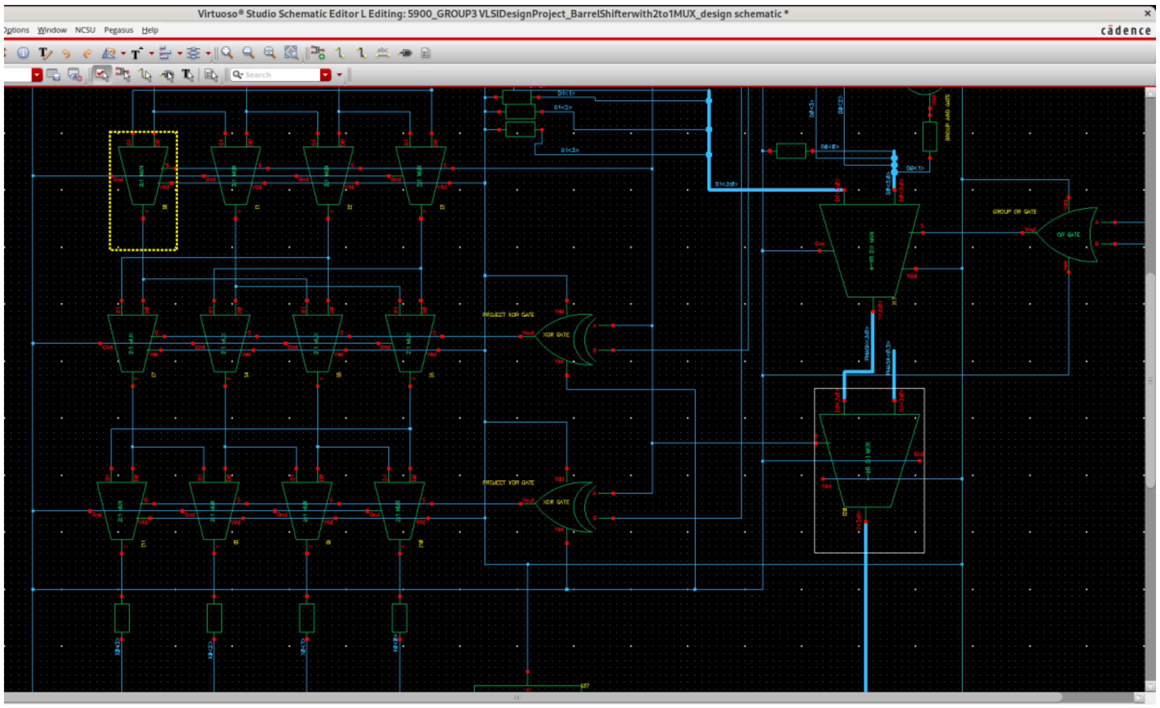

Virtuoso® Studio Schematic Editor L Editing: 5900_GROUP3 VLSIDesignProject_BarrelShifterwith2to1MUX_design schematic *
Options Window NCSU Pegasus Help
cādence
PROJECT XOR GATE
XOR GATE
GROUP OR GATE
OR GATE

Figure above shows a close view of the Barrel Shifter

A two-stage network of 2x1 multiplexers is used to create the 4-bit Barrel Shifter, which applies logical shifts to a 4-bit input bus. Every stage that selects sends either the original bit or a version that has been shifted by one or more positions, controlled by the shift selection lines. The input bits were initially fanned downward into a standard array of mux cells. Without adding more storage components, the circuit may achieve shifts of 0–3 places by compounding these stages. The schematic's OR-gate components on the right combine the mux outputs and drive the final.

SHIFT_OUT<3:0> bus. In order to maintain the schematic's readability and to align with the highest-level structured datapath interface, each signal is connected as buses. Since each 2x1 multiplication unit can be easily repeated in rows as well as columns using shared energy rails and a short local connection, this mux-oriented topology is very regular, transfers neatly into static CMOS in the SkyWater130 PDK, and is layout-friendly.

EXPERIMENT SECTION (LAYOUTS SECTION)

Typical Simulation Parameters

- Technology: digital SkyWater130 CMOS devices.
- Supply: Vdd = 1.8
- Global GND reference
- Clock: The objective is functional verification rather than performance stress testing; every universal CLK signal is represented as an ideal square wave with a period selected to be longer than the anticipated worst-case propagation delay.
- Signal width: Four-bit vectors (e.g., INST<3:0>, R1_OUT<3:0>, ALU_OUT<3:0>) are used to mimic instructions, data, and internal buses.
- Stimulus: To modify patterns without altering the core architecture, inputs are applied via DC voltage sources, pulse generators, or switch-controlled drivers.

To ensure that variations in performance are caused by design architecture rather than testbench variance, specific values are kept constant.

COMMAND STATE MACHINE LOGIC

Sequencing is handled by the command state machine and related decoder:

- WRITE and READ register functions
- SL, SR (barrel shifter), ADD, SUB, AND, OR, and XOR are operations performed via ALU.

To focus on these features, the FSM testbench:

- Uses a series of 4-bit opcodes on INST<3:0> to drive the instruction register. A specific instruction (such as "write R1," "add R1 and R2," and "shift left R2") is associated with each opcode.
- Permits the FSM to move through all necessary states (decode, read, execute, write-back) by applying the global CLK while maintaining the stability of each instruction for multiple cycles.
- Primarily keeps an eye on control outputs, such as multiplexing select signals, shift command bits, ALU operation choose lines, and register enabled, Datapath values are not yet checked.

Confirming that the FSM generates the proper sequence of control signals to implement read, write, arithmetic, logic, and shift operations for every command while ensuring that no contradictory permits are claimed is the overall goal for these trials.

DATAPATH ELEMENTS (REGISTERS, PRODUCT PATH, MULTIPLEXING)

The pair of operand registers, the product/write-back register, the necessary multiplexing, and the logical connections to the FSM are the main topics of the datapath experiments.

Tests for Register and Register Files

Tests using the Register_file block and 4-bit registers:

- The write-data bus with recognized 4-bit patterns or drive D<3:0>.
- Toggle the EN or one-hot register enable lines in accordance with timeframes (e.g., load R1, preserve R2, update R3) that replicate the desired FSM control.
- Only when enabled are active may synchronously state updates be confirmed by applying the global CLK and observing Q<3:0> and Qbar<3:0>.

The goal is to confirm that the product register and the two operand registers function as edge-triggered storage components, allowing for proper read-modify-write sequencing in subsequent full-processor tests.

HIERARCHICAL DESIGN METHODOLOGY - FROM THE CELLS TO BLOCKS

A strategy that reflects the hierarchical structure:

Basic standard cell style blocks

Testbenches are utilized at the lowest level to verify how primitives that resemble standard cells behave:

Logical functions: the fundamental pull-up/pull-down structures are implemented by NAND, NOR, NOT, and XOR gates.
Adders: The ripple-carry adder uses single-bit complete adder slices.
The storage components that make up registers are D latches, D flip-flops, and enabled D flip-flops.
Any pass-gate or mux transistor architectures utilized in multiplexers and shifters are referred to as pass/transfer elements.

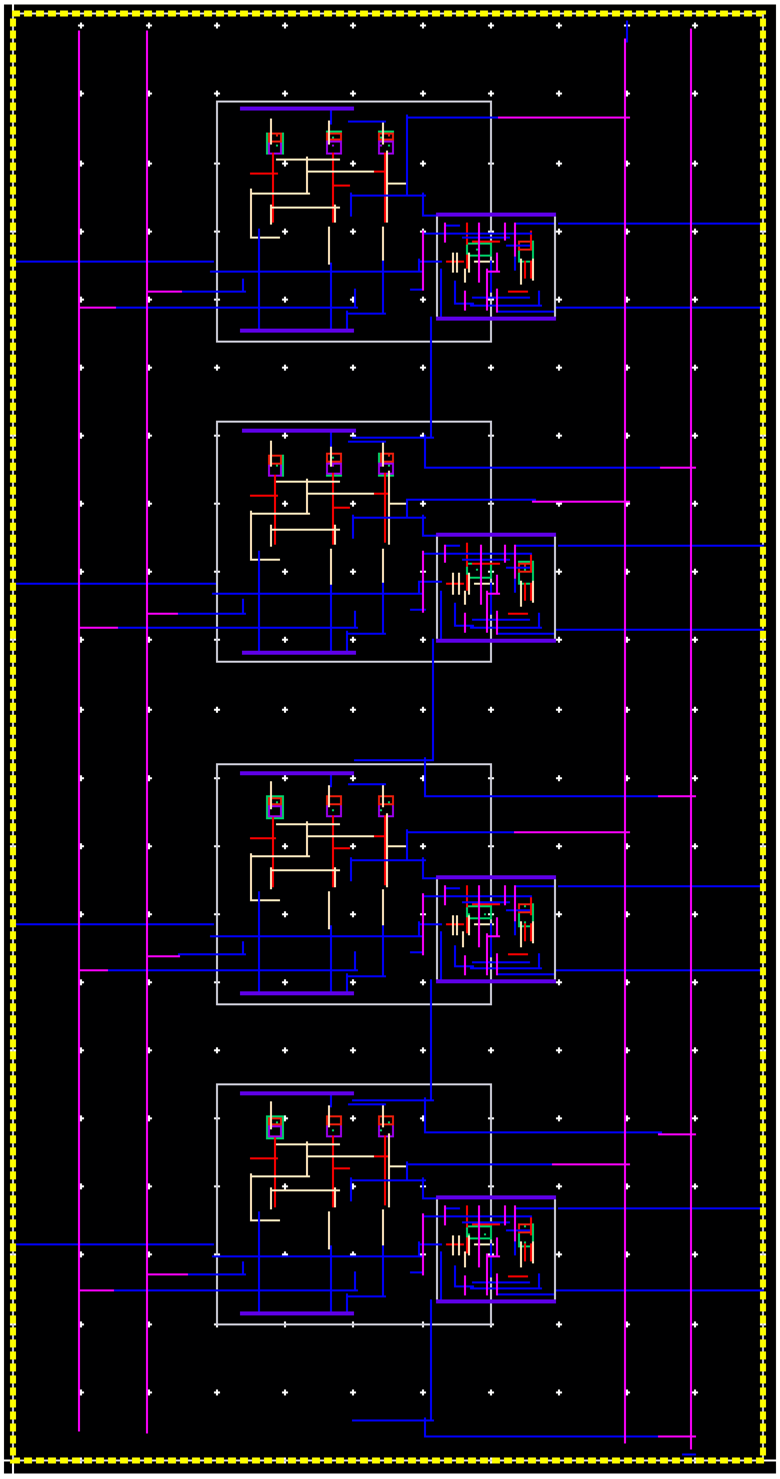
Logic and Data blocks constructed from standard cells

Higher-level blocks are used once the basic elements have been verified:

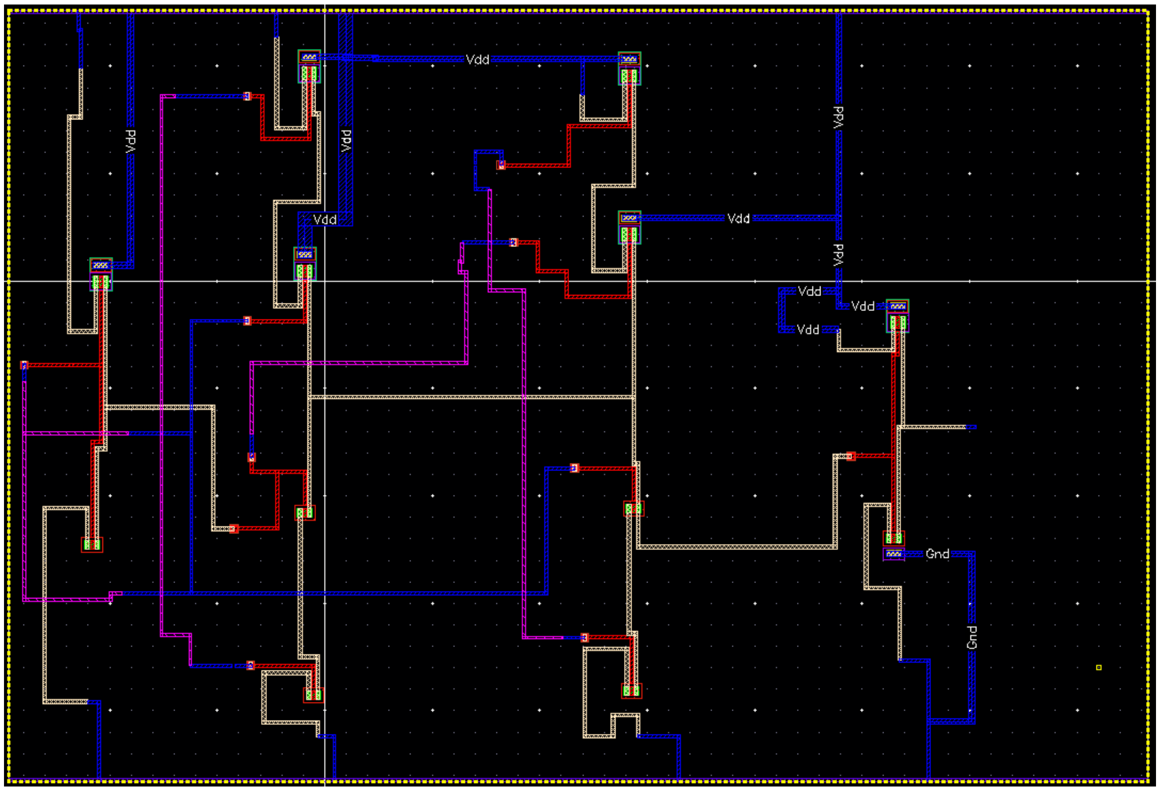
- Logic gates and chained full-adder cells are used to construct the adder and ALU.
- Barrel Shifter constructed on a standard 2x1 mux cell grid.
- State-Machine and Decoder subsections constructed using flip-flops and logic trees.

Every one of these blocks has a testbench that stimulates control signals and common and unusual input patterns. In order for the assembled modules to be securely recycled at the top level, the design goal is to ensure that they act as idealized versions of the textbook components they represent.

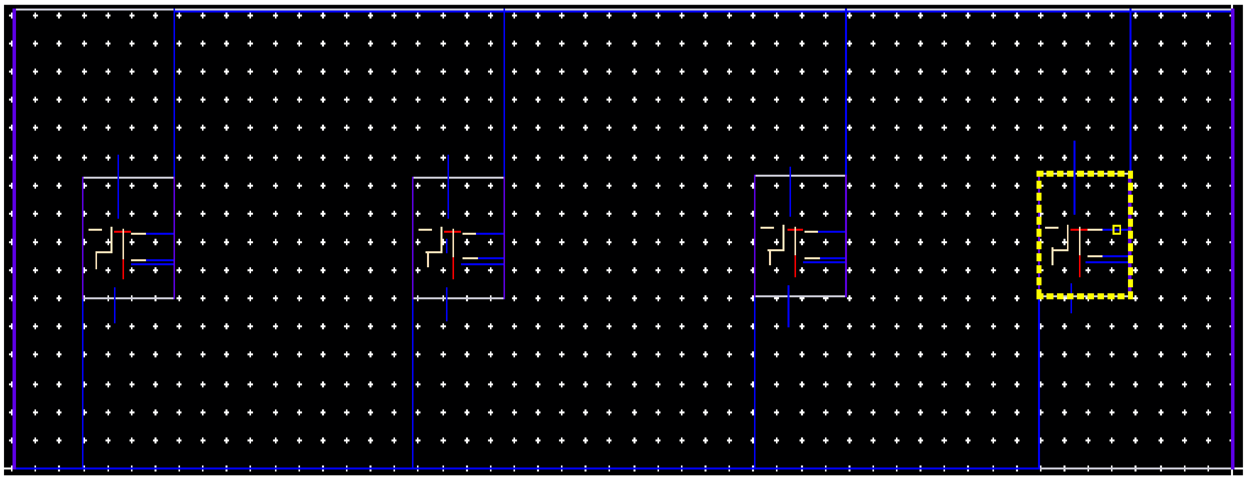
LAYOUTS SECTION



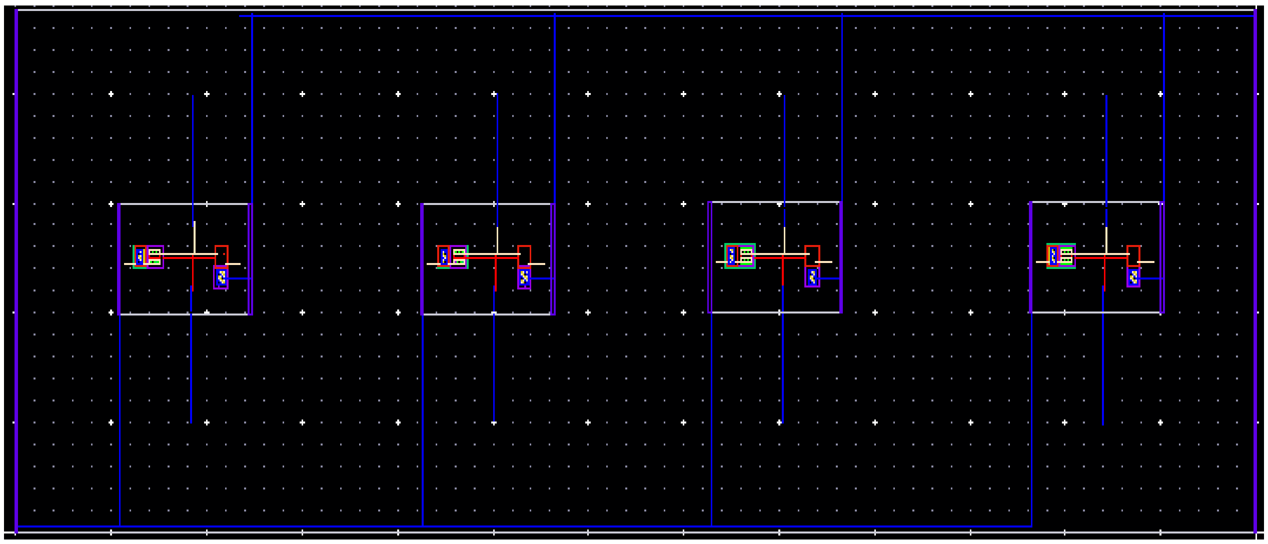
The diagram above shows the layout of the 4-Bit Register



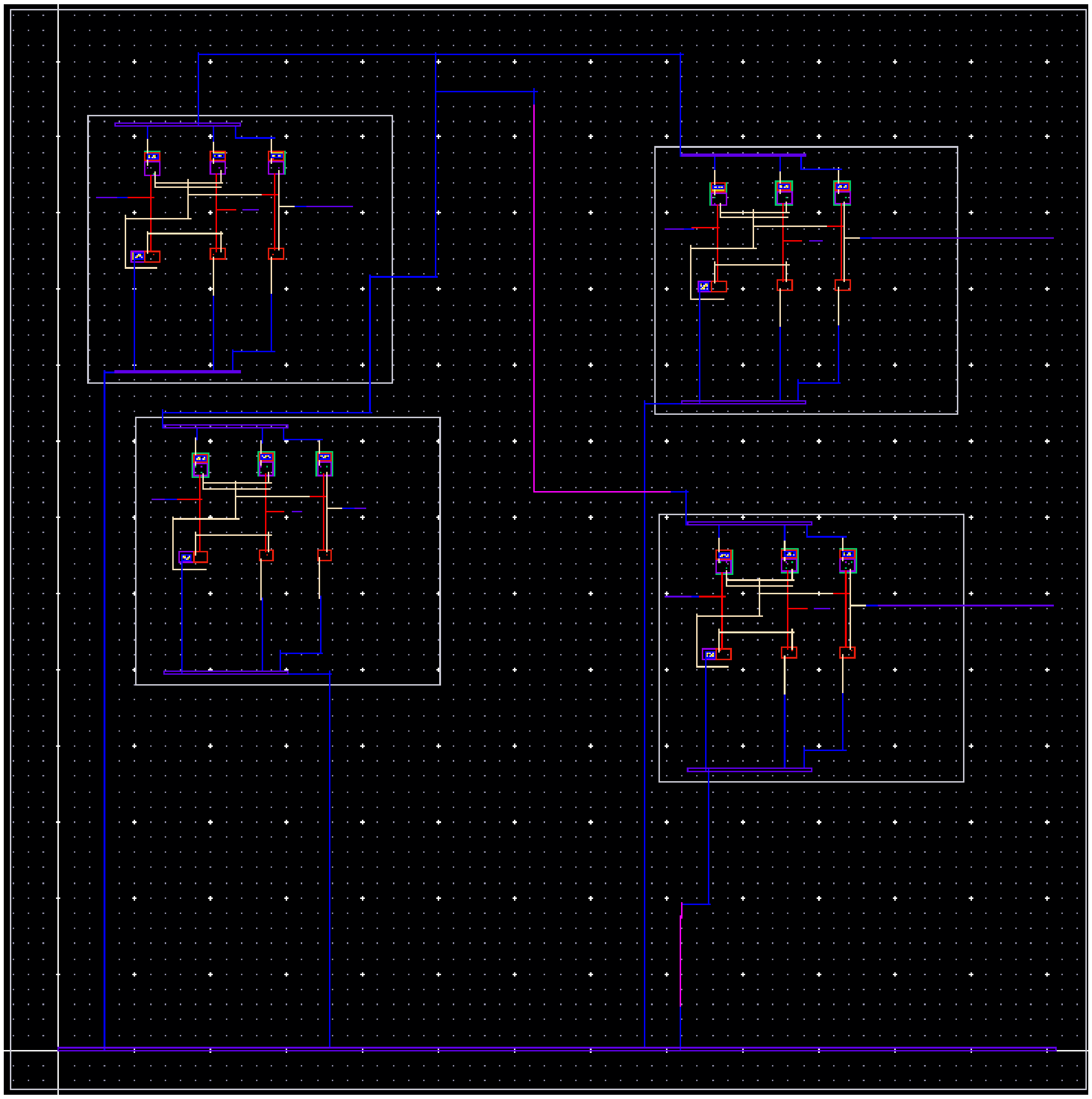
The diagram above shows the CMOS_2to1_MUX_Layout



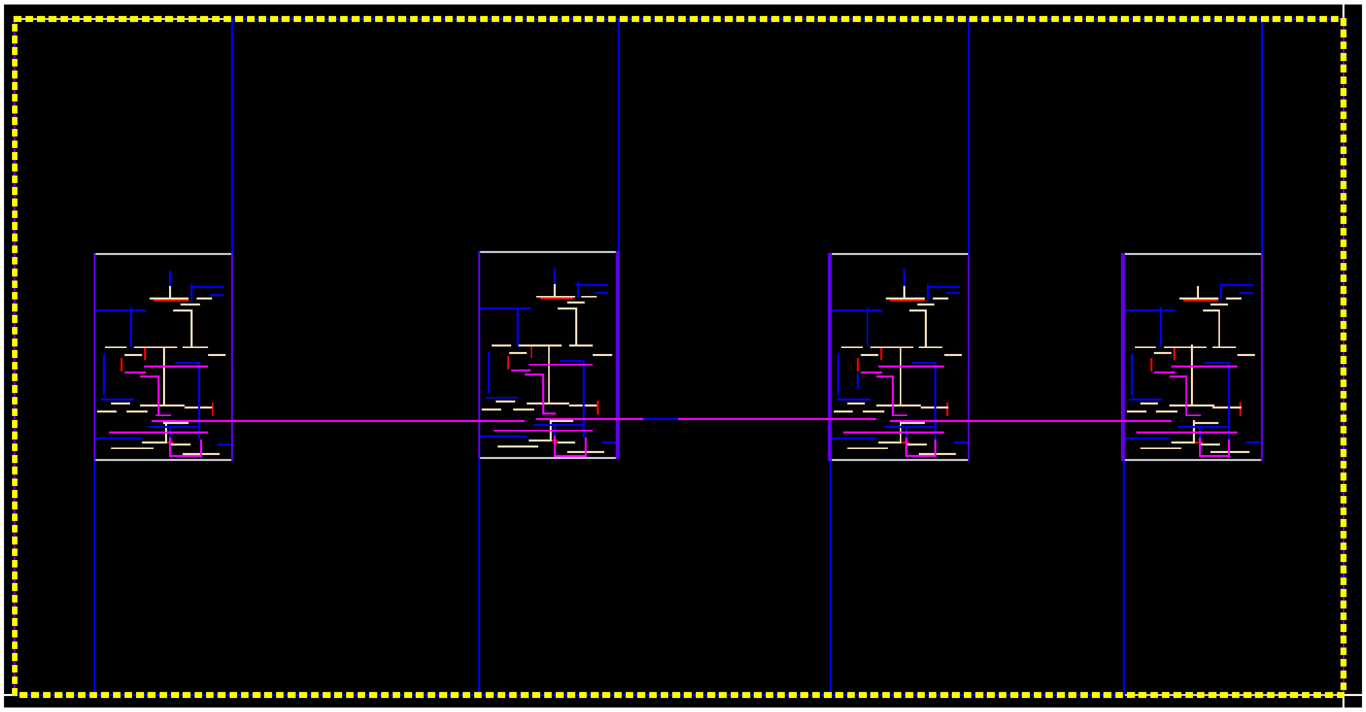
The diagram above shows the layout of the CMOS_NOR_Gate

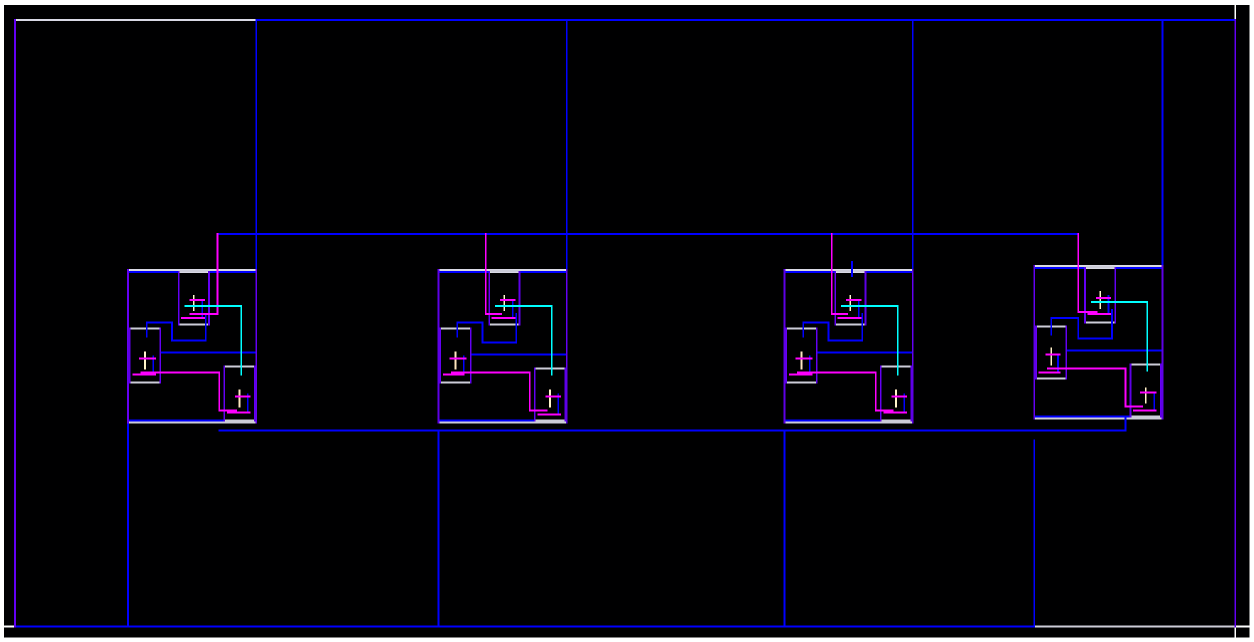


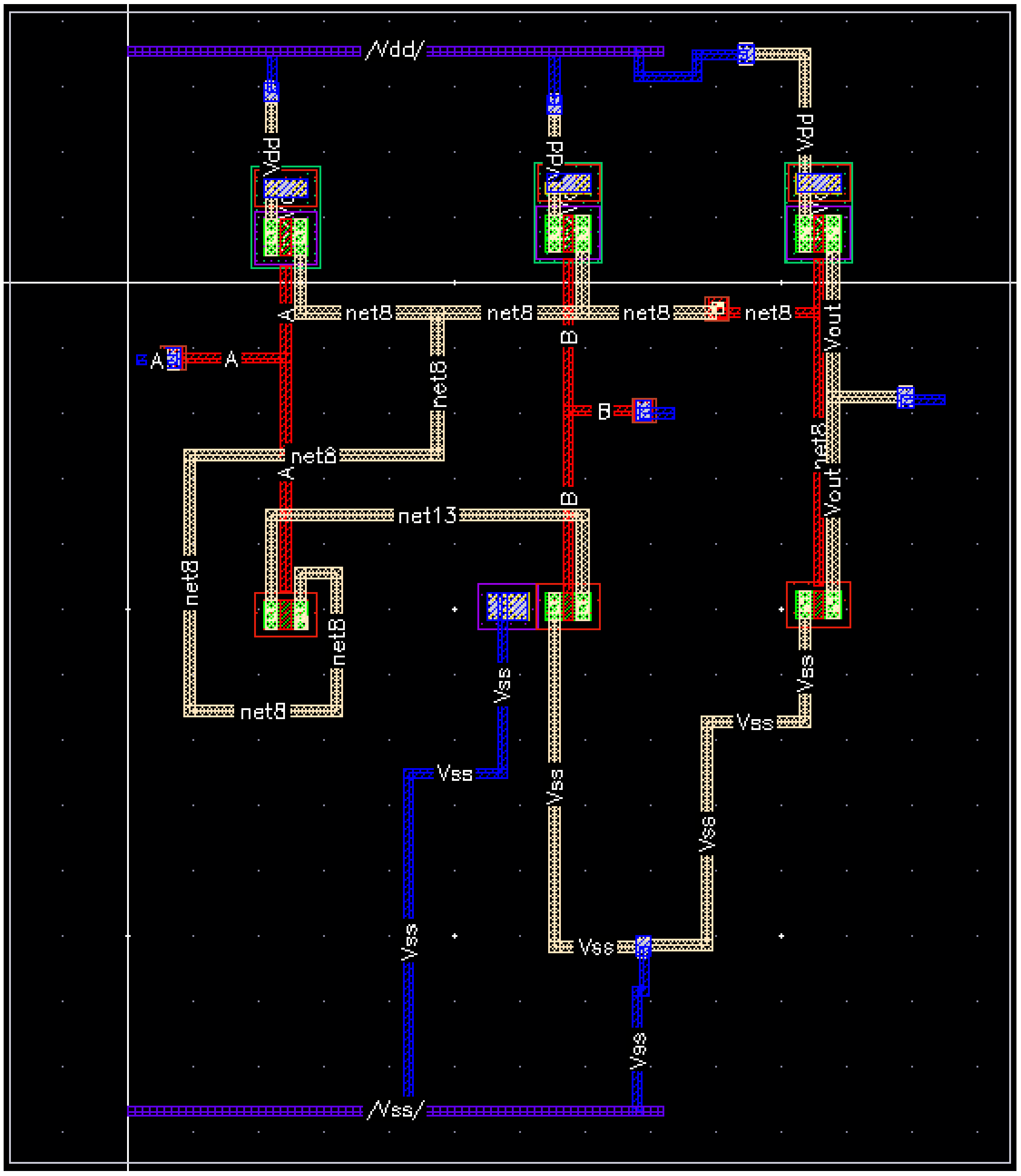
The diagram above shows the layout of the CMOS_4BIT__INVERTER



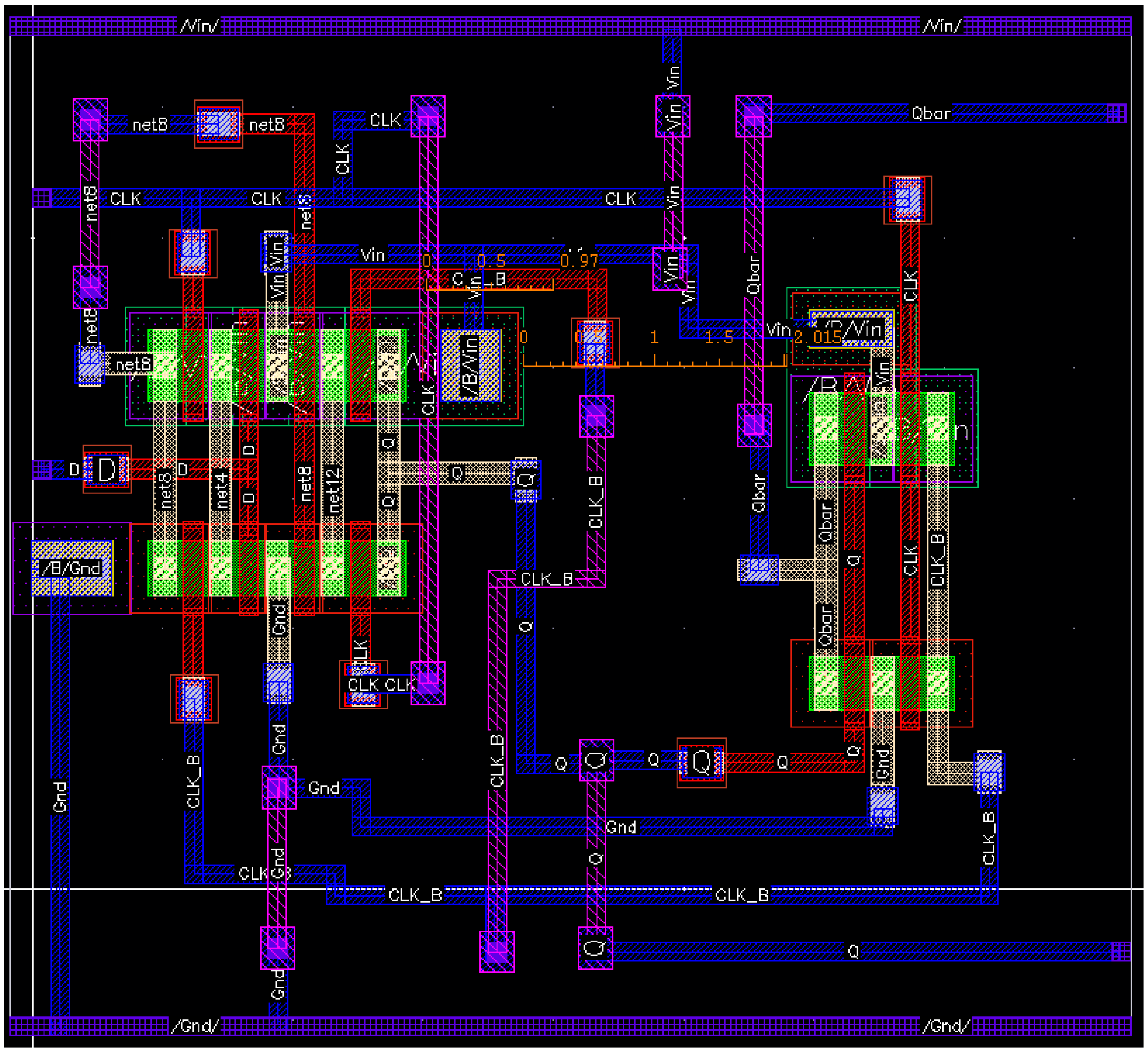
The diagram above shows the layout of the CMOS_4Bit_AND_Gate


The diagram shows the layout of the 4_Bit_2x1_MUX

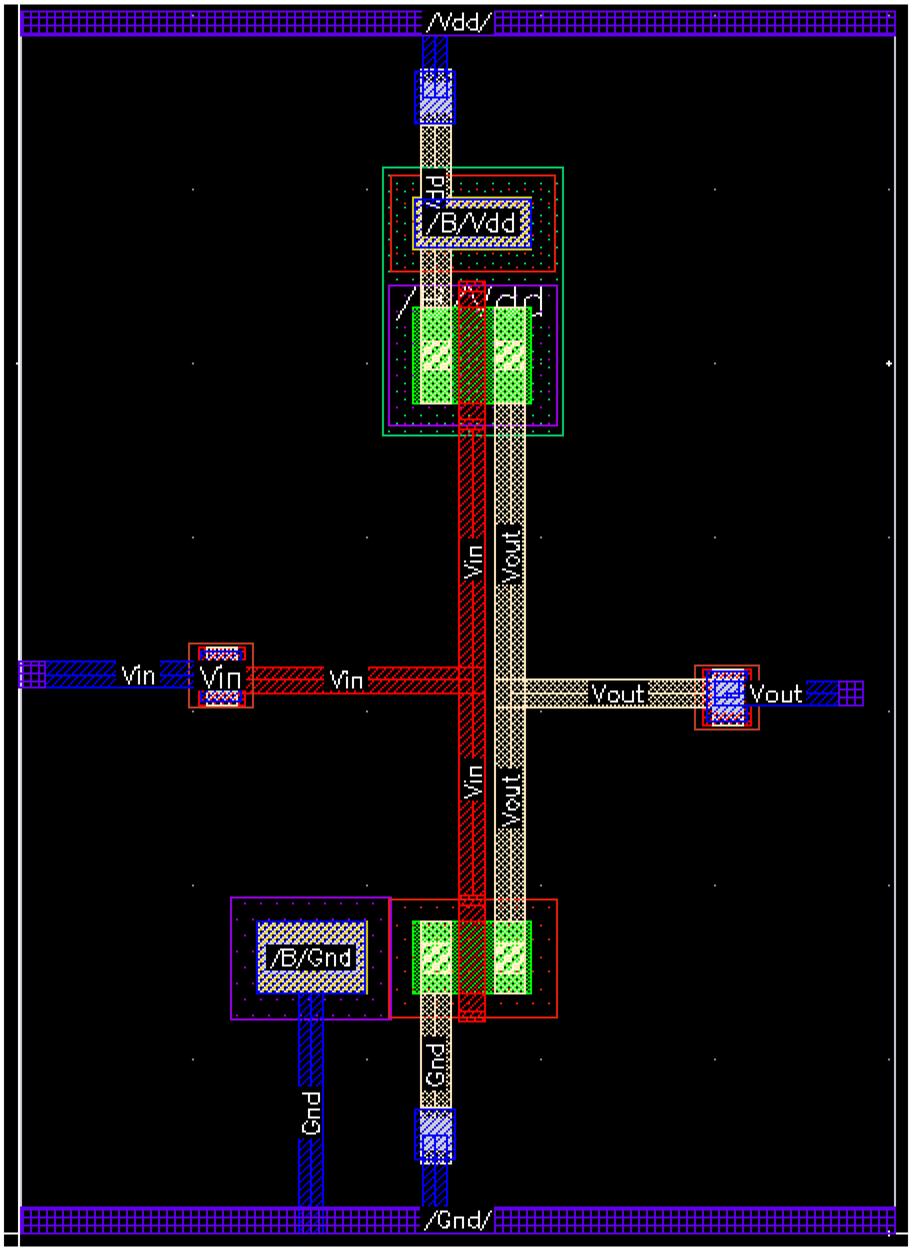

The diagram shows that the layout of the 4bit_4_to_1MUX Layout



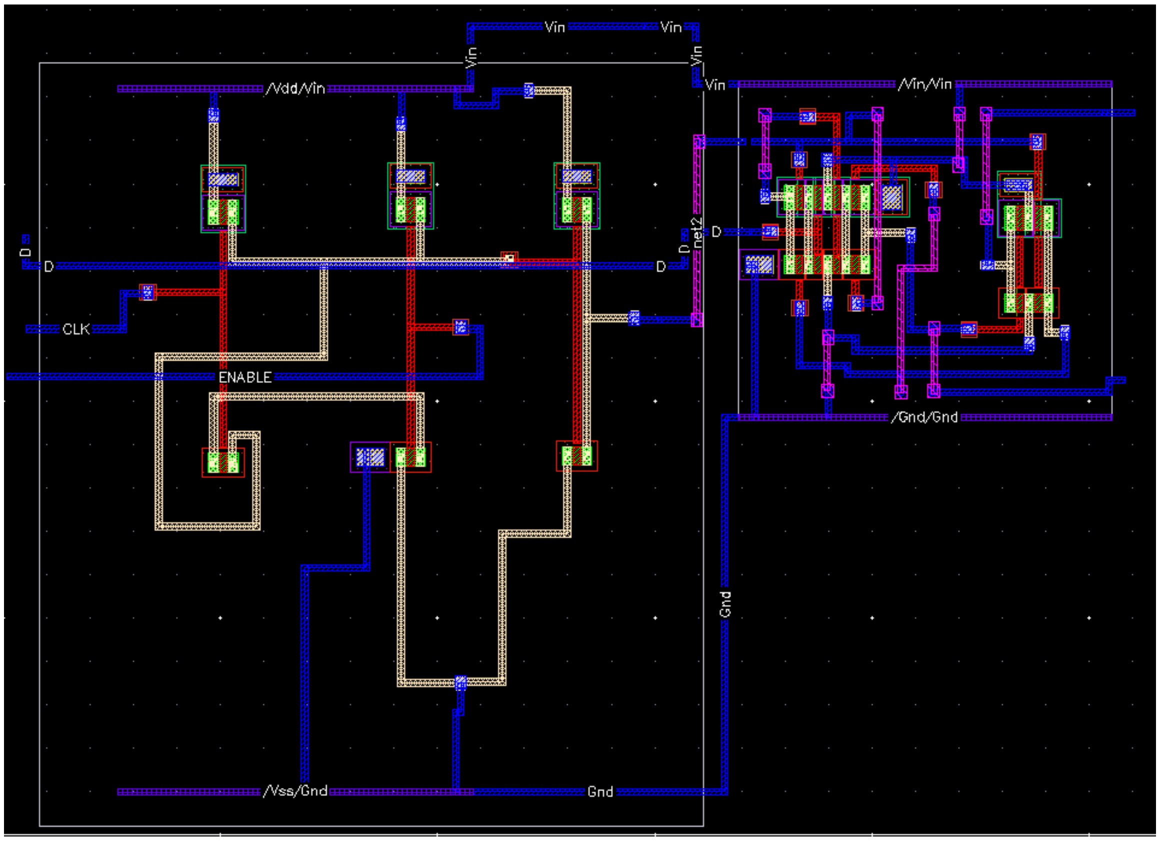
The diagram above shows the CMOS_AND Layout



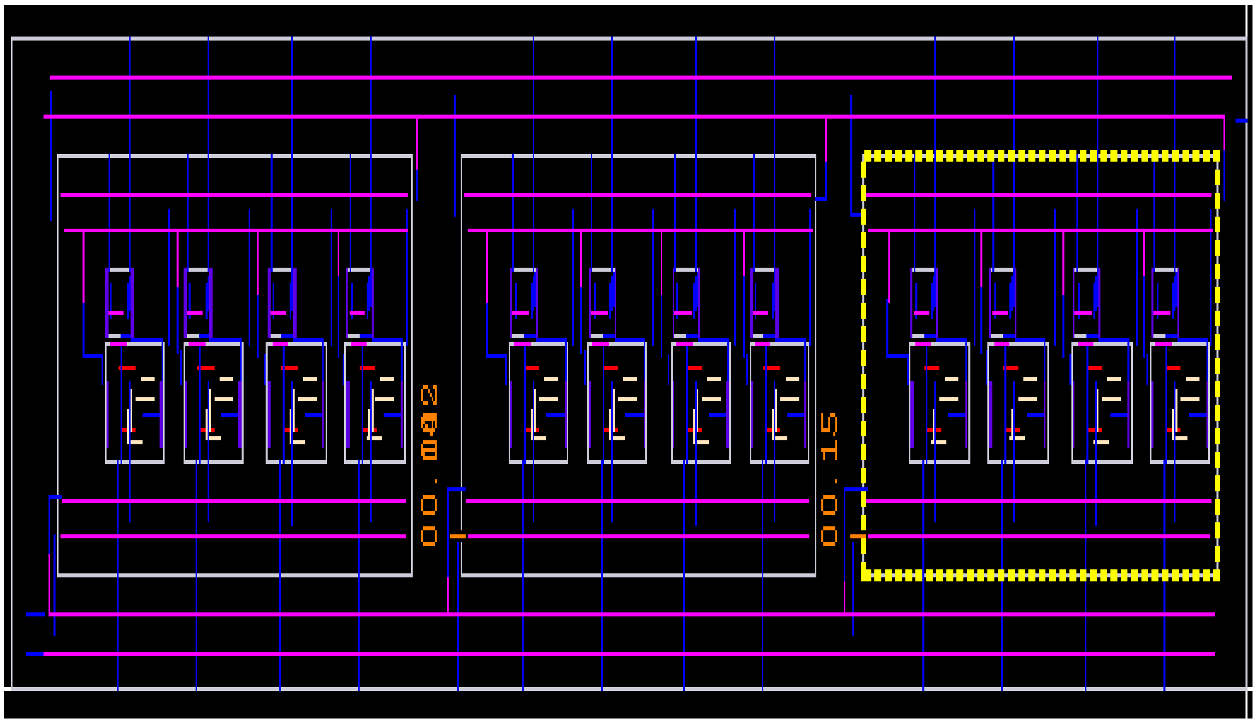
The diagram above shows the layout of the D_Flip_Flop



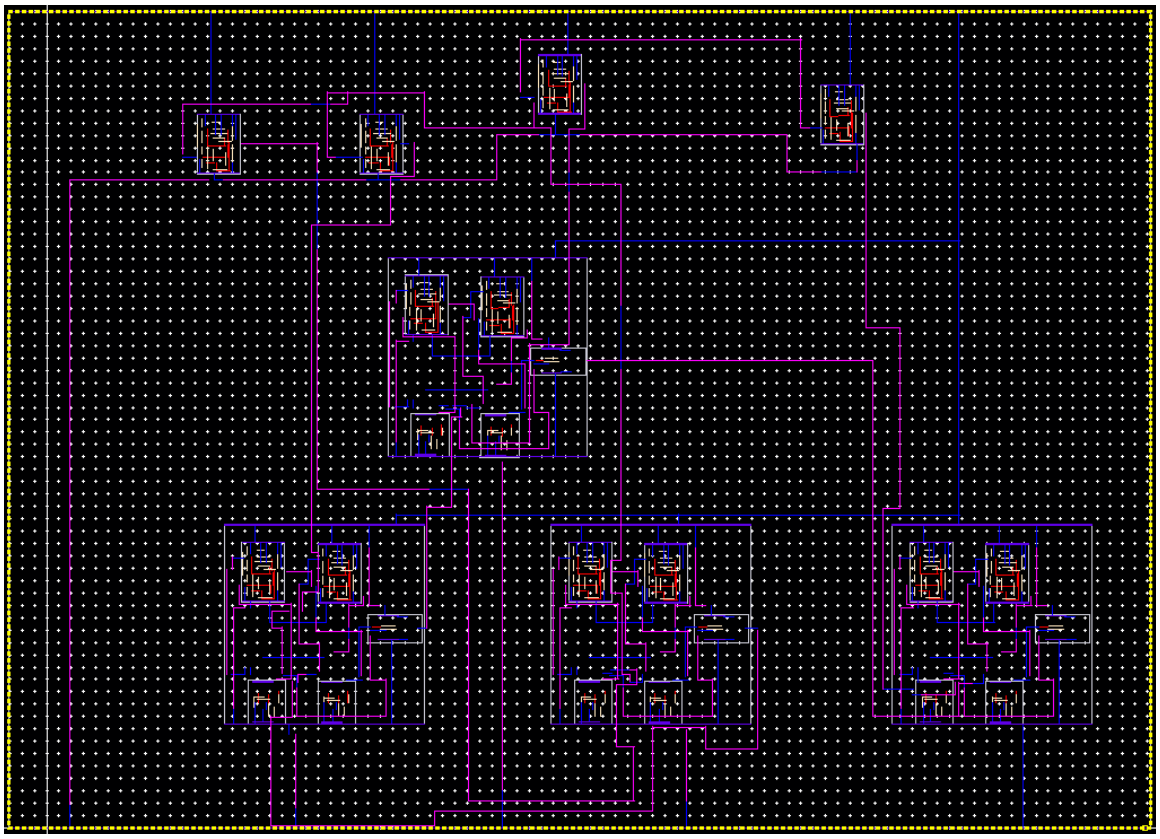
The diagram above shows the layout of the CMOS_INVERTER



The diagram above shows the layout of the Enable_D_Flip_Flop



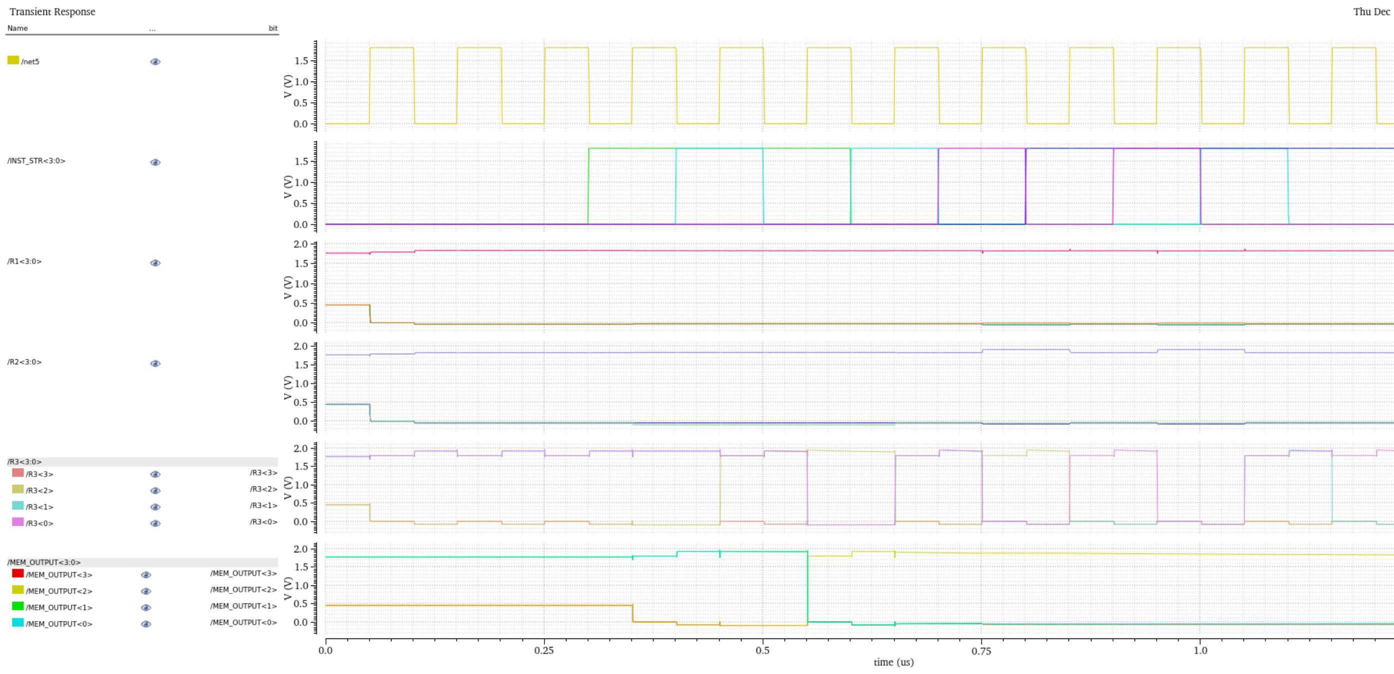
The diagram above shows the layout of the Register_File



The diagram above shows the layout of the CMOS_4Bit_RippleCarryAdder

RESULTS AND DISCUSSION SECTION

4-BIT RISC PROCESSOR SIMULATION RESULTS AND FINDINGS



Processor Simulation with R1:0010 R2:001

Machine Code	Instr	Expected R3 Value	Simulation R3 Value	Expected MEMOUT	Simulated MEMOUT Value
0000	ADD(Instruction enable is low and R1 and R2 are initialized)	xxxx	xxxx	xxxx	xxxx
0000	ADD(Instruction enable is low and R1 and R2 are initialized)	xxxx	0011	xxxx	0011
0000	ADD	0011	0011	xxxx	0011
1000	WRITE	xxxx	0101	0011	0011
1100	SL	0100	0100	0011	0011
1000	WRITE	xxxx	0001	0100	0100
0100	READ	0100	0100	0100	0100

0001	SUB	0101	0101	0100	0100
0010	AND	0000	0000	0100	0100
0011	OR	0011	0011	0100	0100
0110	SR	0000	0000	0100	0100

The diagram above shows the processor simulation results.

The diagram above displays the processor's top-level transient response for a brief instruction sequence. The global clock is at the top of the trace, followed by the 4-bit instruction stream INST_STR<3:0>, the register file outputs R1<3:0>, R2<3:0>, and R3<3:0>, and the external MEM_OUTPUT<3:0> bus. Only one register changes state during each write-back phase, and the register contents only update on rising clock edges when the instruction field changes in discrete stages.

The outcomes verify the coherence of the Datapath and control logic. While R3 is updated repeatedly in accordance with a series of load and compute instructions, R1 and R2 retain their values for extended periods of time. The memory/output bus shows that the write-back multiplexing and register enable are properly synchronized with the instruction timing when it follows the value of the presently selected register and stays steady between programmed updates. The conclusion that the synchronous register file and FSM satisfy the functional targets for top-level integration is supported by the absence of bogus intermediary levels or inadvertent simultaneous updates.

Conclusion

The primary objectives of the 4-bit RISC CPU design project were accomplished. As a three-person team, we created a processing architecture, designed and tested the schematics, and even created layouts that the SkyWater130 PDK could build. A small register file, a ripple carry ALU supporting ADD, SUB, AND, OR, and XOR, a barrel shifter for logical shift left and shift right, a command state machine, and a multiplexer network for operand and write back routing are all included in the final circuit. The primary blocks have been verified to be physically consistent with their schematics using iterative Pegasus DRC, LVS, and ERC checks, making them ready for complete chip floor planning and verification.

One of the most important aspects was the extremely effective use of a tiered hierarchical VLSI approach: logic cells and flip flops were used first, followed by registers, adder slices, shifter cells, decoder segments, and finally the Datapath and control. Finding and debugging problems at the appropriate level of abstraction was extremely feasible thanks to that framework. In addition to the clear ownership of the primary blocks, the shared accountability for verification and integration, the use of Google Drive, Google Chat, and a centralized master directory for all design files, which greatly reduced integration errors and version conflicts, the team improved their working methods through the project.

The value of early and stepwise DRC/LVS, which was crucial in identifying open pins, misaligned metals, and enclosure violations that differed from top level integration, the importance of interface discipline (regular pin naming, orientation, and bus conventions), which made multi-designer integration possible, and the role of excellent communication and file management as the primary technical skills required for the project's smooth operation aside from the managerial tasks are some of the key lessons learned.

In many respects, the work paves the door for a number of possible future paths. A timing-driven design with post-layout delay extraction, additional research into faster adder/shifter topologies, and possibly the addition of a straightforward pipeline would constitute the implementation side. As a long-term objective, the flow may eventually be partially automated using HDL-based design, synthesis, and experimental location and route. Overall, the project has provided a solid foundation for more intricate integrated circuit designs and has been a great hands-on introduction to contemporary VLSI tools and workflows.

11. References

[1] Weste, N. H. E., & Harris, D. M. (2011). *CMOS VLSI design : a circuits and systems perspective* (4th ed.). Addison-Wesley.

[2] Harris, Sarah ; Harris, David. *Digital Design and Computer Architecture, RISC-V Edition*. Morgan Kaufmann, 2021.