Project Plan report format – Your first page MUST match this format

/10

Name: Manav Kulabkar
Unityid: mmkulabk
StudentID:. 220540228

Summary Risk Plan: I designed my ALU unit and simulated on paper and excel sheets using rough timing diagrams and logic circuits according to my algorithms.
I had to verify the functioning of my FSM by comparing the modelsim output and the rough timing diagrams I created.

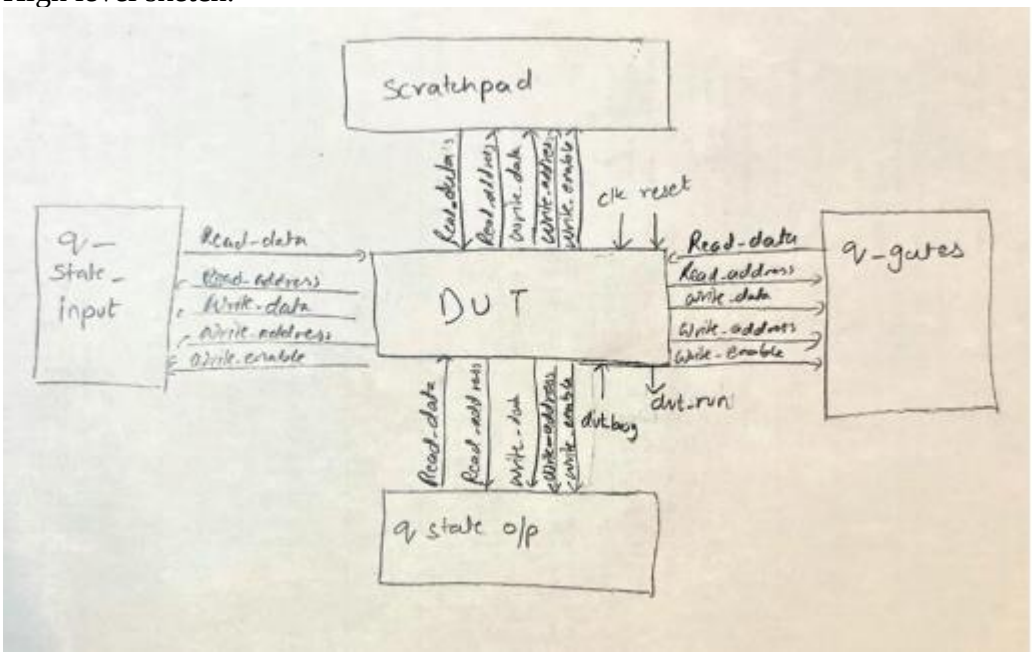
Schedule:
20th Nov: Design and timing diagram completed
24th Nov: Functioning code with all test cases passed.
25th Nov: Synthesis completed
26th Nov: Report completed

Brief Description of Mode of operation, including selected algorithms:

- In this design, we're combining complex floating point numbers stored in different SRAMs. In the initial process of the design, we need to extract the number of qubits and the number of operator matrices. We will obtain the number of elements and the number of matrices to be dealt with.

- The real part and the imaginary part of the complex numbers which are of 64 bits each are separated and are sent to the ALU for the main objective of this project. Real part of State Input data gets combined with real part of Operator Matrix and the same goes for the imaginary part after which the numbers are concatenated to get the original 128 bit hexadecimal value which will be written to the Output SRAM or the Scratchpad SRAM depending on the number of matrices.

High level sketch.



Final Project Report First Page. Must match this format (Title)

Name: Manav Kulabkar
Unity id: mmkulabk
StudentID:. 200540228

Delay (ns to run provided provided example). Clock period: 50ns # cycles": 2934 cycles	Logic Area: 113822.7305 (um^2)	1/(delay.area) ($ns^{-1}.um^{-2}$) $1.75718276*10^{-7}$ or 0.000000175718276
Delay (TA provided example. TA to complete)		1/(delay.area) (TA)

Abstract

This project involves the development of a quantum emulator with complex floating-point values. The emulator supports qubits ranging from 1 to 4, and multiplication of matrices ranging from 2 to 21. Quantum computing leverages qubits, capable of existing in multiple states simultaneously through superposition. This unique property enables quantum computers to outperform classical counterparts in certain computations. Qubits can also be entangled, establishing instantaneous connections between their states and facilitating intricate interactions. The project underscores the quantum computational potential, where n qubits have the capacity to represent 2^n states. In the architecture at hand, the utilization of four Static Random-Access Memories (SRAMs) is thoughtfully allocated for distinct roles: two serve the purpose of reading input data, one is dedicated to inscribing the final output, and a fourth—aptly named the Scratchpad SRAM—functions as an auxiliary memory reservoir, effectively contributing to a reduction in overall area usage. Floating-point computations are facilitated by the DW_fp_mac module from the Synopsys designware library. This design can perform superscalar execution only by adding extra number of MAC units in the design. Despite the design's possession of four Multiply-Accumulate (MAC) units and an intentionally parsimonious allocation of registers, the present constraint revolves around the ability to read only one data point per cycle, a limitation dictated by the intricacies of SRAM operations. Nevertheless, the design navigates this challenge with finesse, employing a strategic incorporation of cycle stalling techniques. This tactical approach not only ensures adherence to the specified project requirements but also serves as a testament to the design's commitment to achieving less area along with optimal efficiency and precision in the derivation of results.

Project Title

Manav Kulabkar

Abstract

Quantum computing leverages qubits, capable of existing in multiple states simultaneously through superposition. This unique property enables quantum computers to outperform classical counterparts in certain computations. Qubits can also be entangled, establishing instantaneous connections between their states and facilitating intricate interactions. The project underscores the quantum computational potential, where n qubits have the capacity to represent 2^n states. In the architecture at hand, the utilization of four Static Random-Access Memories (SRAMs) is thoughtfully allocated for distinct roles: two serve the purpose of reading input data, one is dedicated to inscribing the final output, and a fourth—aptly named the Scratchpad SRAM—functions as an auxiliary memory reservoir, effectively contributing to a reduction in overall area usage. Floating-point computations are facilitated by the DW_fp_mac module from the Synopsys designware library. This design can perform superscalar execution only by adding extra number of MAC units in the design. Despite the design's possession of four Multiply-Accumulate (MAC) units and an intentionally parsimonious allocation of registers, the present constraint revolves around the ability to read only one data point per cycle, a limitation dictated by the intricacies of SRAM operations. Nevertheless, the design navigates this challenge with finesse, employing a strategic incorporation of cycle stalling techniques. This tactical approach not only ensures adherence to the specified project requirements but also serves as a testament to the design's commitment to achieving less area along with optimal efficiency and precision in the derivation of results.

1. Introduction

This project delves into the intricate world of quantum emulation, focusing on the development of a sophisticated quantum emulator designed to handle complex floating-point values. The versatility of the emulator is highlighted by its support for qubits ranging from 1 to 4 and matrix multiplications spanning an ambitious spectrum from 2 to 21.Navigating the challenges posed by 128-bit hexadecimal inputs from SRAMs, our design intricately dissects these numbers. The first step involves partitioning them into 64-bit segments, distinguishing the real and imaginary components of complex numbers. Subsequently, the 64-bit floating-point values undergo further deconstruction into fraction and exponent parts, with the all-important Most Significant Bit (MSB) determining the sign bit.

Data flow within the emulator is facilitated through two pivotal SRAMs—State_input and Q_gates. The initial address of the State_input SRAM emerges as a linchpin, housing critical information for computing matrix dimensions and kickstarting the Finite State Machine (FSM). This initialization cycle sets the stage for the FSM and its associated control lines, serving a dual purpose by initializing counters for subsequent calculations and steering the flow of the FSM.

The FSM itself, at the heart of our design, boasts 13 states (excluding the initial reset state), choreographed by five select lines. While limiting states to 13 may seem like a constraint, this intentional choice streamlines functionality and enhances comprehensibility. The incorporation of color-coded control lines within the FSM adds a layer of simplicity, facilitating the recognition and understanding of each state's unique role.

In summary, our quantum emulation design, while grappling with intricate complexities, not only meets stringent technical requirements but also provides a transparent and accessible framework for comprehending

fascinating intricacies of quantum emulation.

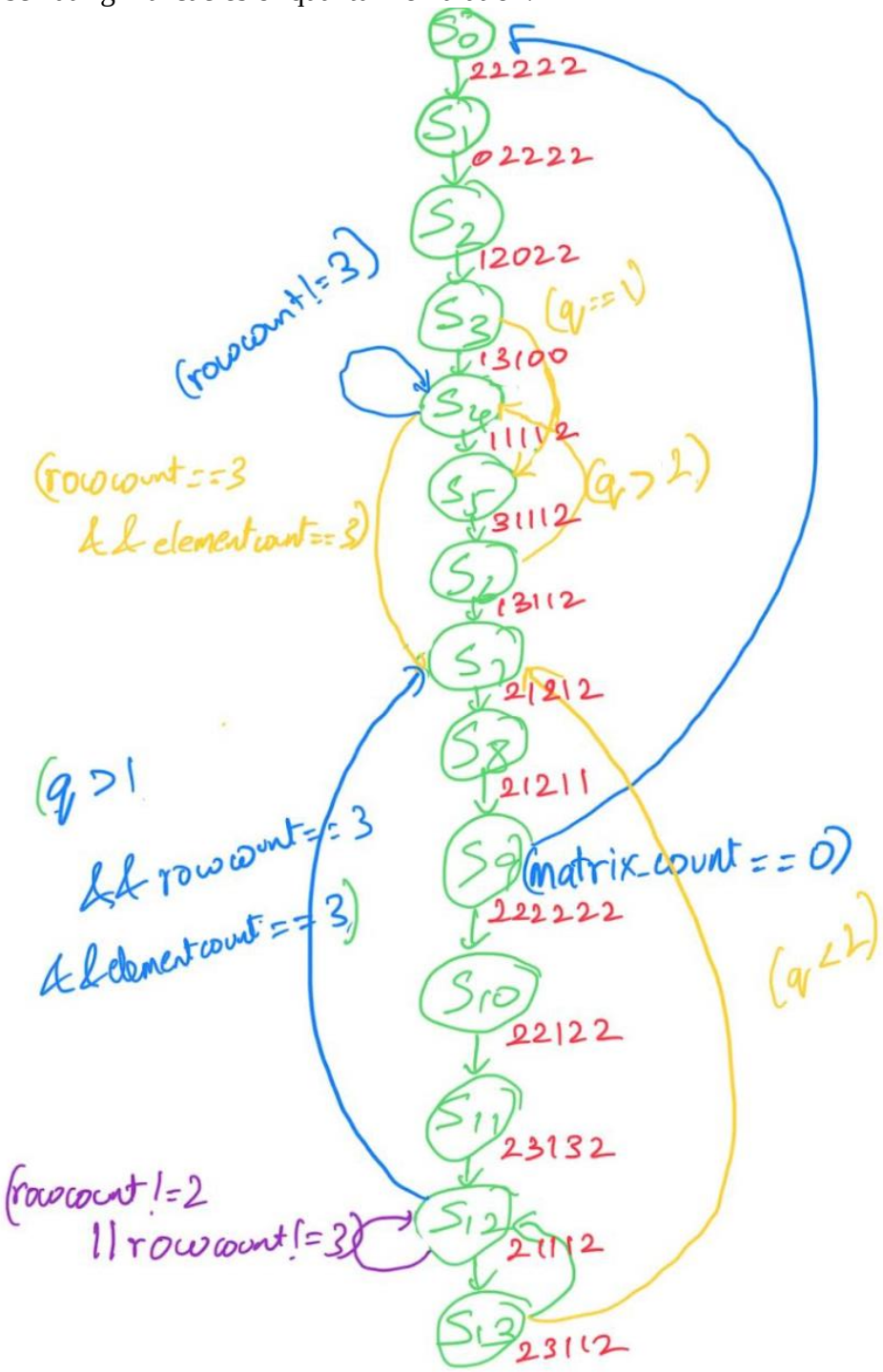


Figure: FSM of the current design.

General algorithm:

- Initialise q and m values and the rowcount, elementcount and the matrixcount.
- Start reading the number of address equal to 2^q from State Input and Qgates.
- Decrement the counters and each cycle there is a new read data pass it to the ALU for computation.
- After eacg cycle if the rowcount becomes ‘1’, turn on the scratchpad write enable or the output write enable depending on m.
- Write will be issued from 0^{th} address and everytime there is a new write, write address has to be incremented by 1 to avoid any RAW hazard.
- After element count == 0, issue read from scratchpad if m>1 and stall input from qgates and decrement the value of matrixcount.
- Start reading from scratchpad and qgates at the same cycle and repeat the process of computing and writing.
- If matrixcount ==1 write in output sram.
- Else write in scratchpad and keep repeating the process until matrixcount == 1.

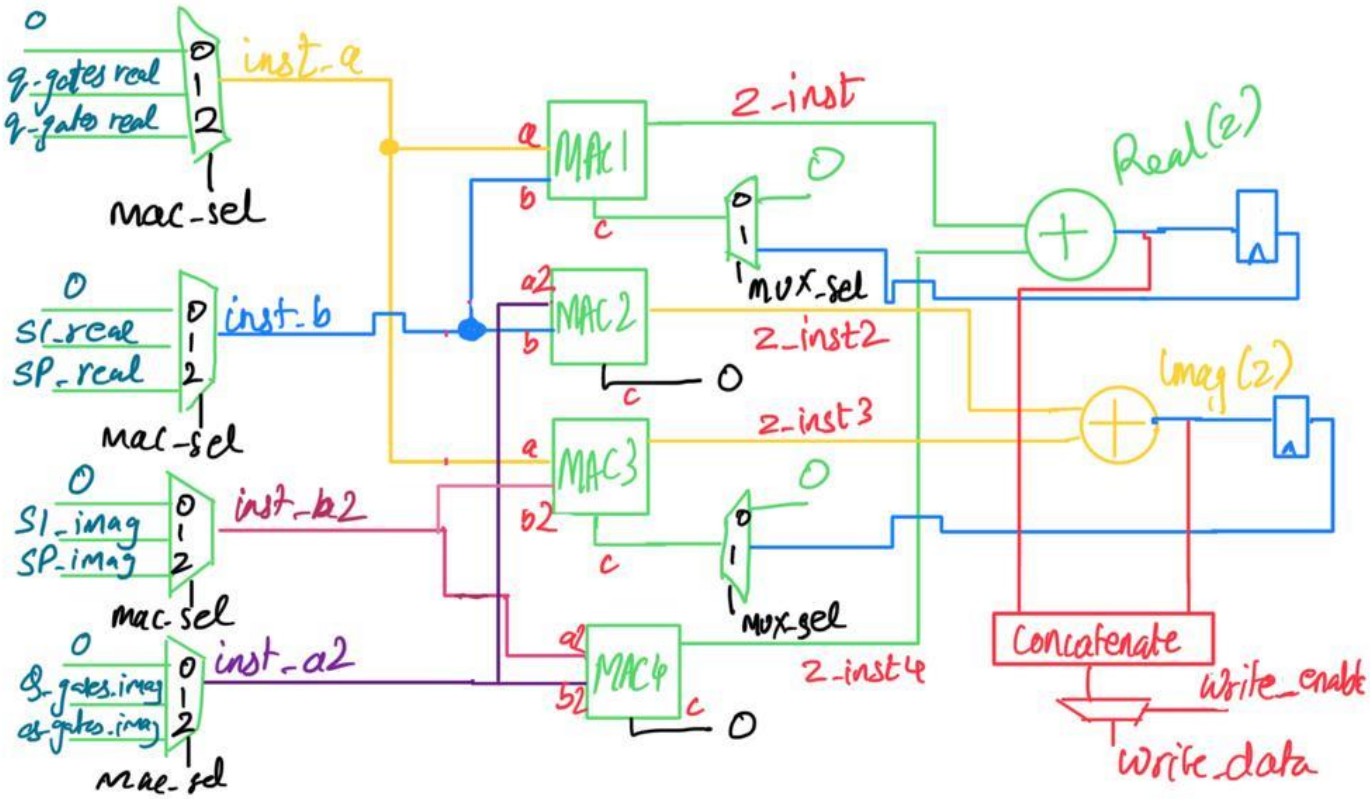
- **Summary of Results Achieved**

- The outcomes of this project underscore the effectiveness of strategic design choices in achieving notable results:
- **Optimized Design Area**: One of the primary achievements is the significant reduction in the design's overall footprint. This accomplishment is particularly noteworthy when compared to scenarios where arrays or buffers were employed without the inclusion of a scratchpad. The judicious integration of a scratchpad not only aids in minimizing area usage but also enhances the overall efficiency of the design.
- **Enhanced Interaction with SRAMs**: The decision to incorporate a scratchpad has proven instrumental in optimizing the design's interaction with multiple Static Random-Access Memories (SRAMs) concurrently. This simultaneous interaction in the same cycle, facilitated by the scratchpad, contributes to improved efficiency and responsiveness in handling data inputs.
- **Dynamic Area Reduction with Decreasing Clock Period**: An intriguing observation arises from the relationship between the clock period, area utilization, and SLACK. The progressive reduction in the clock period correlates with a proportional decrease in the design's footprint. This not only emphasizes the dynamic nature of area optimization but also underscores the critical importance of meeting SLACK conditions for sustained improvements.
- In essence, the achieved results not only validate the efficacy of incorporating a scratchpad for area reduction but also shed light on the dynamic interplay between clock period adjustments, area utilization, and design efficiency. These findings contribute to a nuanced understanding of the delicate balance required for optimal quantum emulator performance.

Report contents

2. Micro-Architecture

- Hardware Algorithm of ALU and data flow



- There are two important select lines here, mac_sel and mux_sel.
- Mac_sel decides which input is combining with q_gates. It is either the State Input or the scratchpad(if m>1).
- The data has already been partitioned into real and imaginary before feeding into the ALU.
- Output of MAC1 and MAC4 are responsible for generating the real part of the result
- Output of MAC2 and MAC3 are responsible for generating the imaginary part of the result.
- mux_sel decides when the new data gets accumulated with the old data that is coming from the flipflop.
- Write enable decides when to write the accumulated data to the sram and just before that, we concatenate the real and imaginary part of the result.

3. Interface Specification

Detailed Description of Top-Level Interface

- The top-level interface plays a crucial role in connecting our quantum emulator with Static Random-Access Memories (SRAMs). We manage this interaction by passing address and data lines as parameters to the DUT module, ensuring a smooth flow of information.

Interaction with SRAMs

- The communication between the DUT and SRAMs is well-organized. For SRAMs dedicated to reading (State Input and Q_gates), we use only the necessary read lines. In the case of the Scratchpad SRAM, which serves both reading and writing, both read and write lines are employed. This ensures efficient use of excess memory while keeping data accessible for computation.

- Interestingly, the Output SRAM, used solely for writing, requires no read lines in the DUT. This highlights a straightforward, one-way interaction where the emulator sends output directly to this SRAM.

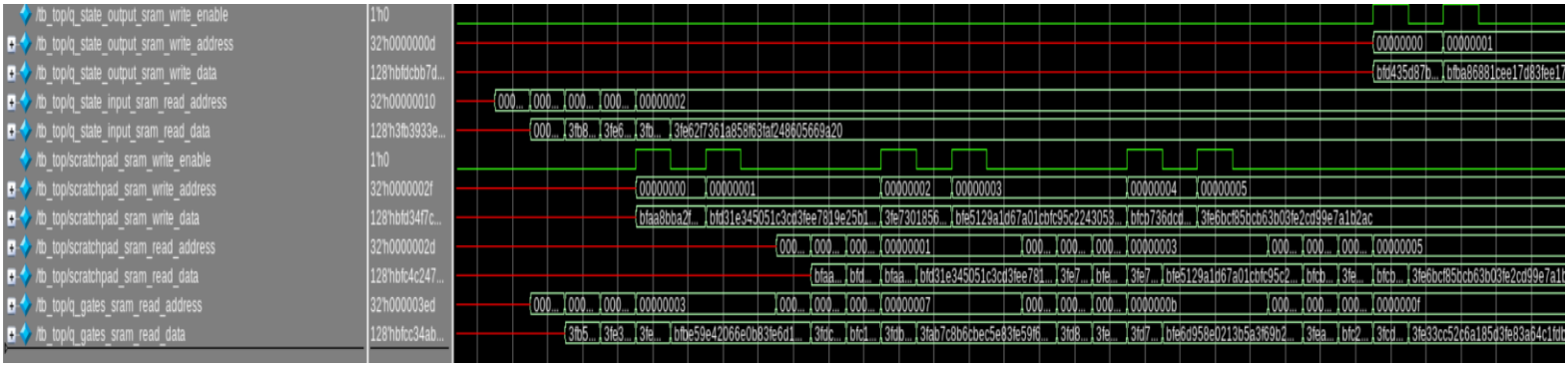
- In essence, our simple yet effective top-level interface ensures a practical and streamlined connection between the quantum emulator and SRAMs, meeting the unique needs of each memory unit.

Table for top level interface

Sr No	Signal Name	Width	Function
1	q_state_output_sram_write_enable	1	Sending the write enable as high indicating new data to be written to the SRAM
2	q_state_output_sram_write_address	32	Contains the address of the data to be written to output SRAM
3	q_state_output_sram_write_data	128	Contains the data to be written in the SRAM
4	q_state_input_sram_read_address	32	Contains the address of the data to be read from state input
5	q_state_input_sram_read_data	128	Contains thethe data to be read from state input
6	scratchpad_sram_write_enable	1	Sending the write enable as high indicating new data to be written to the Scratchpad SRAM
7	scratchpad_sram_write_address	32	Contains the address of the data to be written to
8	scratchpad_sram_write_data	128	Contains the data to be written in the SRAM
9	scratchpad_sram_read_address	32	Contains the address of the data to be read from scratchpad
10	scratchpad_sram_read_data	128	Contains the data to be read from scratchpad

11	q_gates_sram_read_address	32	Contains the address of the data to be read from q_gates
12	q_gates_sram_read_data	128	Contains the data to be read from q_gates

- Timing diagram of top level interface



4 Verification

Ensuring the correctness of our quantum emulator is foundational to its reliability. Our approach to verification is multifaceted, encompassing key aspects to validate the functionality and integrity of the design:

- **State Transitions and Select Lines:** Verification begins by rigorously confirming the accuracy of state changes and their associated select lines. This involves meticulous tracking of alterations in all registers and wires linked to control lines and states. By scrutinizing these changes, we validate that the quantum emulator progresses through its states as intended, crucial for the overall fidelity of the system.

- **Scratchpad and Output SRAMs:** Verifying the write operations to the Scratchpad and Output SRAMs involves a hands-on inspection of waveforms. Manually scrutinizing the data and addresses sent to these SRAMs ensures that information is correctly stored and retrieved. This manual verification process serves as a robust check against inadvertent errors in data transmission.

- **Timing Diagram Consistency:** To further bolster our verification efforts, a timing diagram, initially outlined on paper, is compared with the corresponding output from ModelSim. This visual alignment serves as an additional layer of confirmation, providing confidence in the temporal coherence of our quantum emulator's operations.

- **MAC Computation Accuracy:** The heart of our design lies in the Multiply-Accumulate (MAC) units. Ensuring the correctness of computations involves a meticulous process of converting signal radices and manually cross-verifying calculated values. This granular verification step adds an extra layer of assurance regarding the precision and reliability of the quantum computation performed within the emulator.

- In summary, our verification approach encompasses not only the overarching state transitions but also delves into the specifics of memory interactions, timing consistency, and the accuracy of core computations. This comprehensive methodology ensures a robust and validated quantum emulator, instilling confidence in its ability to faithfully execute quantum computations.

5. Results Achieved

- Time period: 50 ns
- Area: 113822.7305
- # of cycles (test 6): 2934

6. Conclusions

- In the culmination of this project, our endeavors have yielded a robust and capable quantum emulator, showcasing several key achievements and successful implementations:
- **Matrix Multiplication Mastery:** The quantum emulator, a product of meticulous design and execution, stands as a testament to its prowess in multiplying matrices of substantial scale. Handling matrices with elements ranging from 4 to 64, each containing complex numbers, underscores the versatility and computational might embedded within our design.
- **Precision with dw_fp_mac Module:** The integration of the dw_fp_mac module has proven instrumental in achieving unparalleled accuracy in complex calculations. Leveraging this module, we've successfully navigated the intricacies of floating-point operations, ensuring that the quantum emulator delivers precise results even in the face of formidable computational challenges.
- **Synchronized SRAM Interfacing:** Interfacing with different SRAMs has been a resounding success, demonstrating the emulator's adaptability in handling diverse memory operations. This accomplishment extends to simultaneous and distinct operations on multiple SRAMs, showcasing the versatility of our design in orchestrating complex data interactions seamlessly.
- **Synthesis Error Mitigation:** The synthesis phase presented its own set of challenges, notably including issues such as latches and the utilization of the negedge clock. Tackling these challenges head-on, our approach involved a judicious restructuring of the logic and code. This not only resolved synthesis errors but also contributed to a more streamlined and efficient design.
- In conclusion, the successful design and implementation of our quantum emulator represent a significant milestone in the realm of quantum computing. The emulator's prowess in handling intricate matrix multiplications, its precision with complex calculations, adept interfacing with diverse SRAMs, and the resolution of synthesis challenges collectively affirm the viability and efficacy of our quantum computational model. This project not only advances our understanding of quantum emulation but also lays a foundation for further exploration and innovation in this cutting-edge field.