Quantum Computing Emulator
Final Project Report

Name: Ashwin Sarathi Krishnan
Unityid: asarath
StudentID: 200542907

Delay (ns to run provided provided example): 64360
Clock period: 38
cycles": 1694

Logic Area: 80559.43(um^2)

Memory: N/A

1/(delay.area) ($ns^{-1}.um^{-2}$) = 1.9287 x 10^-10

Delay (TA provided example. TA to complete)

1/(delay.area) (TA)

Abstract
The goal of this project is to design a quantum computing emulator. The various operator matrices are multiplied with the initial state vector to produce a final output state vector. These calculations can involve real as well as imaginary values. I solved this problem by building an 8 state FSM. The system is designed to parse the initial state vector along with the operator matrices through 2 MAC units and an adder unit and the results were written to a scratchpad sram. This scratchpad sram held the result of each intermediate operator matrix calculation and the final result was written to the output sram. This FSM managed to successfully simulate all the provided test cases. The synthesis was successful and the system was optimized to a final clock of 38 ns, with an area of 80559.4303 um^2. Its final performance was 5184792045 (ns x um^2).

Quantum Computing Emulator Design

Ashwin Sarathi Krishnan

Abstract

The goal of this project is to design a quantum computing emulator. The various operator matrices are multiplied with the initial state vector to produce a final output state vector. These calculations can involve real as well as imaginary values. I solved this problem by building an 8 state FSM. The system is designed to parse the initial state vector along with the operator matrices through 2 MAC units and an adder unit and the results were written to a scratchpad sram. This scratchpad sram held the result of each intermediate operator matrix calculation and the final result was written to the output sram. This FSM managed to successfully simulate all the provided test cases. The synthesis was successful and the system was optimized to a final clock of 38 ns.

1. Introduction

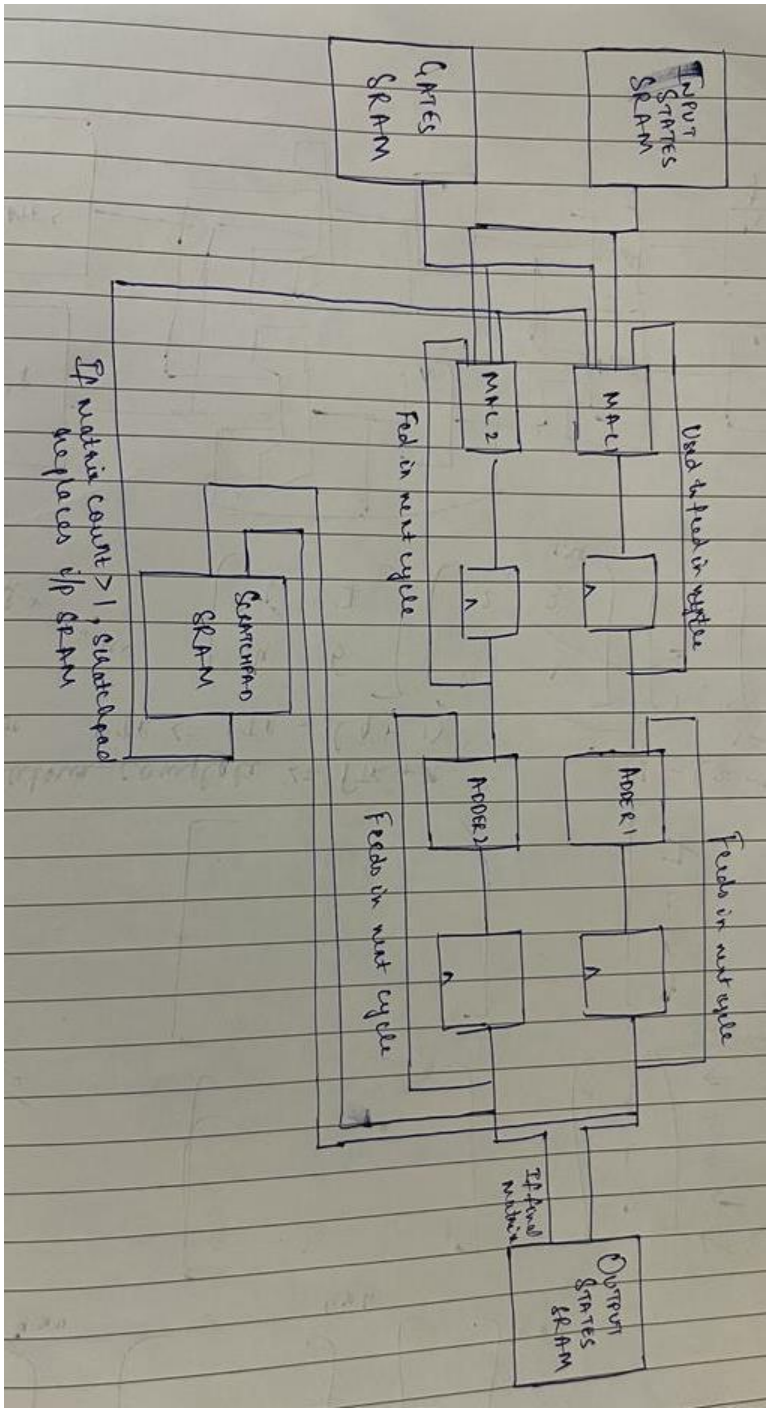
The hardware being designed here is a quantum computing emulator. The emulation of this construct is done by designing hardware that is capable of reading an initial state matrix and operator matrices. Using the FSM algorithm, the hardware successfully generates the corresponding output state matrix, which have been verified using a comprehensive set of test cases. The rest of this report will go over the design micro-architecture, interface specification, verification, the results achieved and the conclusion arrived at from this project.

2. Micro-Architecture

The algorithm is as follows:

- The state input sram address 0 is read to find out the number of states in the matrix and total number of operator matrices. This is also used to generate the total number of inputs we are expecting from the gates sram.
- The input sram is then incremented.
- The data at gates sram address 0 and input sram address 1 is fed into the two MACs. The data is split into 64 bits each and fed into the macs to provide the first stage of outputs.
- These outputs are then stored in 2 temporary registers and are fed to the accumulate inputs of the MACs. The remaining combination of the 64 bit inputs are fed to the multiply unit of the MAC and the resulting output is the result of one multiplication of the matrices.
- The outputs of this multiplication are fed to two temporary registers, one to store the real part and the other to store the imaginary part.
- Then the next cycle begins and the sram read address and gates read address is incremented. The value stored in the temporary register of the adder is fed as the secondary input.
- This continues till one entire row is read. At this point, the sram read address is reset to 1. The value produced by the adders is concatenated and written to the scratch pad.

- This continues till one entire matrix is read. Then the second input is read from the stored values in the scratchpad instead of the states sram.
- Using the total matrices and current matrix count, it is detected if we are reading the final matrix. If so, the write data is written directly to the final output states matrix instead of the scratchpad.



3. Interface Specification

S No.	Signal Name	Width	Function
1	scratchpad_sram_write_enable_r, q_state_output_sram_write_enable_r	1	Assigns the value of the write enable wire to respective sram
2	q_state_input_sram_read_address_r, q_gates_sram_read_address_r, scratchpad_read_address_r	32	Assigns value of read address to respective sram
3	Scratchpad_sram_write_address_r, q_state_output_sram_write_address_r	32	Assigns write address to respective sram
4	Scratchpad_sram_write_data_r, q_state_output_sram_write_data_r	128	Assigns write data to respective sram
5	inst_a1, inst_a2, inst_b1, inst_b2, inst_c1, inst_c2	64	DW_MAC inputs, where a is multiplier, b is multiplicand and c is addend
6	Inst_rnd_mac	3	Rounding input of MAC
7	Status_inst_mac1, status_inst_mac2	8	MAC output status
8	Inst_a3, inst_a4, inst_b3, inst_b4	64	DW_Adder inputs, where a and b are added together
9	Inst_rnd_adder	3	Rounding input of adder
10	Z_inst_mac1, z_inst_mac2	64	Output of MAC units
11	Z_inst_adder1, z_inst_adder2	64	Output of adder units
12	Current state	4	Describes current FSM state of system
13	Next state	4	Describes state of FSM in the next cycle
14	Set_dut_ready	1	Needed for the handshake with the input sram
15	get_q_m	1	When this signal is 1, the values are split into 2 64 bit fragments, and is used to generate q_states_count, gates_count and total_matrices
16	save_q_m	1	When high, it saves the values of q_states_count, gates_count and total_matrices
17	Q_state_input_read_addr_sel, q_gates_read_addr_sel, scratchpad_read_read_sel	2	Serves essentially as the read address pointer for the respective sram.
18	Q_state_output_write_address_sel	2	Serves as a write address pointer for the output sram
19	Sram_write_enable_sel	2	Used to choose which sram to write to
20	Scratchpad_address_controller	2	Helps modify the scratchpad_sram_write_address as per need
21	Input_sram_sel	2	Used to choose which sram's input to read into the MAC unit as the multiplicand
22	Compute_mac	1	Turns on the mac unit when high
23	Compute_adder	1	Turns on adder unit when high

24	Row_read_complete	1	When high indicates that one full row of the operator matrix has been read
25	Gate_read_complete	1	When high, indicates all the addresses in the gates sram has been read
26	Matrix_read_complete	1	When high, indicates that one entire operator matrix has been read
27	Matrix_count_controller	2	Records the number of operator matrices that have been fully read so far, and when it is greater than 0, multiplicand input comes from scratchpad instead of input gates sram
28	Final_matrix_count	1	High when matrix_count_controller is 1 less than total_matrices, used to indicate that the output now has to be written to output sram instead of scratchpad
29	Temp_addend_control	2	Stores the result of the MAC operation to feed as addend for the next operation
30	Temp_adder_control	2	Stores the result of the adder operation to feed as second input to the next adder operation
31	Compute_complete	1	When high, sets dut_ready_r disabling sram inputs
32	Q_states_count	64	Counts the number of addresses in the initial state vector
33	Gates_count	64	Counts total possible operator matrix addresses
34	Scratchpad_address_counter	64	Holds current sram write address value
35	Matrix_count	64	Records the current matrix count
36	Total_matrices	64	Holds maximum possible matrices
37	Temp_mac1, temp_mac2	64	Holds output of macs to be fed back in a future cycle as addend
38	Temp_adder1, temp_adder2	64	Holds output of adders to be fed back in a future cycle as secondary input

5. Verification

A comprehensive set of test benches were used to test this system. The test benches were generated by have multiple combinations of real and imaginary values sent in from an sram module. The test benches were also made to generate random values of q ranging from 1 to 4 and random values of m ranging from 1 to 20. By means of this, all corner cases were tested and successful simulation shows that the system is truly functional.

6. Results Achieved

- Area = 80559.4303 um^2
- Number of Cycles = 1694
- Total simulation time = 64360
- Clock period = 38ns

- Performance = 5184792045 (ns x um^2)
- 1/ (delay x area) = 1.9287 x 10^-10

7. Conclusions

The result of this project is a quantum computing emulator implemented by means of a series of matrix multiplications. Its functionality has been successfully verified by a thorough set of tests. It has an area of 80559.4303 um^2. With a clock cycle of 38 ns and a total simulation time of 64360 ns for the main test, it produced a performance of 5184792045 (ns x um^2).