Name: Nithin Shyam Soundararajan

Unityid: nsounda

StudentID: 200542849

Delay (ns to run provided example).

Clock period: 40

cycles”: 2572

Logic Area: (um^2)

47574.8983

1/(delay.area) (ns^{-1}.um^{-2})

2.043e-10 ns^{-1}.um^{-2}

Delay (TA provided example. TA to complete)

1/(delay.area) (TA)

Abstract

Unlike traditional computing, quantum computing takes advantage of the quantum phenomena where there are no discrete states (0:OFF and 1:ON), but states which are probabilistic in nature. The value of a qubit is based on the principle of superposition and quantum entanglement, and hence nondeterministic. Quantum gates are used to manipulate qubits whose ways are fundamentally different from logic gates. Its state is known only during measurement.

The work presented is an emulation of a quantum computer in hardware of qubit lengths 1 to 4. The operations performed by quantum gates on qubits are characterized using complex number matrix multiplication. The design employs 2 Floating Point Multiply-And-Add units, one for the calculation of the real part and another for the complex part. A single multiplication involving two complex numbers consumes 2 clock cycles. Control of the matrix multiplications is handled using row and column counters. Computations with multiple operator/gate matrices can be handled. A clock period of 40ns and utilized area of 47574.8 um^2 is achieved.

Quantum Computing Emulator

Nithin Shyam Soundararajan

Abstract

Unlike traditional computing, quantum computing takes advantage of the quantum phenomena where there are no discrete states (0:OFF and 1:ON), but states which are probabilistic in nature. The value of a qubit is based on the principle of superposition and quantum entanglement, and hence nondeterministic. Quantum gates are used to manipulate qubits whose ways are fundamentally different from logic gates. Its state is known only during measurement.

The work presented is an emulation of a quantum computer in hardware of qubit lengths 1 to 4. The operations performed by quantum gates on qubits are characterized using complex number matrix multiplication. The design employs 2 Floating Point Multiply-And-Add units, one for the calculation of the real part and another for the complex part. A single multiplication involving two complex numbers consumes 2 clock cycles. Control of the matrix multiplications is handled using row and column counters. Computations with multiple operator/gate matrices can be handled. A clock period of 40ns and utilized area of 47574.8 um^2 is achieved.

1. Introduction

Quantum computing is a cutting-edge computing technology which outperforms traditional computing by utilizing the principles from quantum physics. In classical computers, data is represented using bits 0 and 1, but in quantum computers, data is stored using quantum bits or qubits which are capable of existing in multiple states at the same time, thanks to the principle of superposition. This enables quantum computers to handle parallelism at levels much larger than classical computers. An application where quantum computers shine is factoring large numbers (which uses Shor's algorithm). Quantum computers manipulate data using quantum gates, which make up the quantum circuits.

The hardware designed in this project simulates a quantum computer. The size of the qubits range between 1 and 4. The states and gates are represented using matrices and their interactions are characterized as matrix multiplications of complex numbers. Each multiplication takes 2 clock cycles. A clock period of 40ns and utilized area of 47574.8 um^2 is achieved.

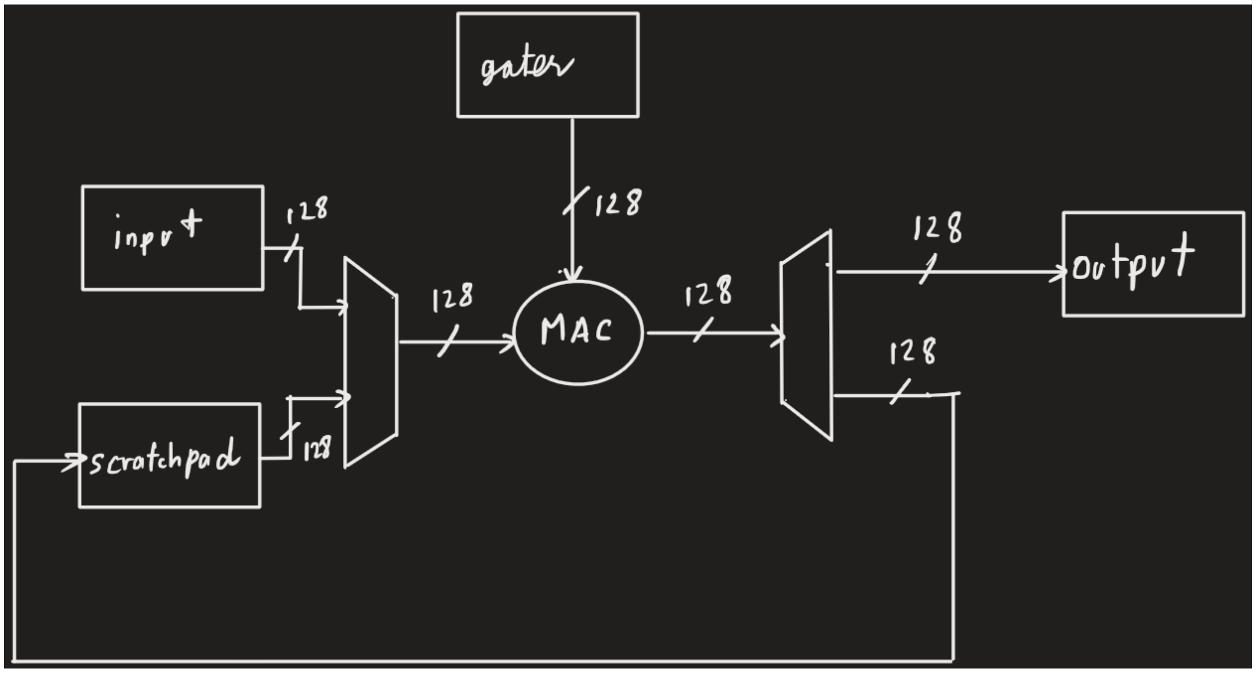
The rest of the report is sectioned as follows: 2. Micro-Architecture, 3. Interface Specification, 4. Technical Implementation, 5. Verification, 6. Results Achieved and 7. Conclusion.

2. Micro-Architecture

There are 4 data stores (SRAMs).

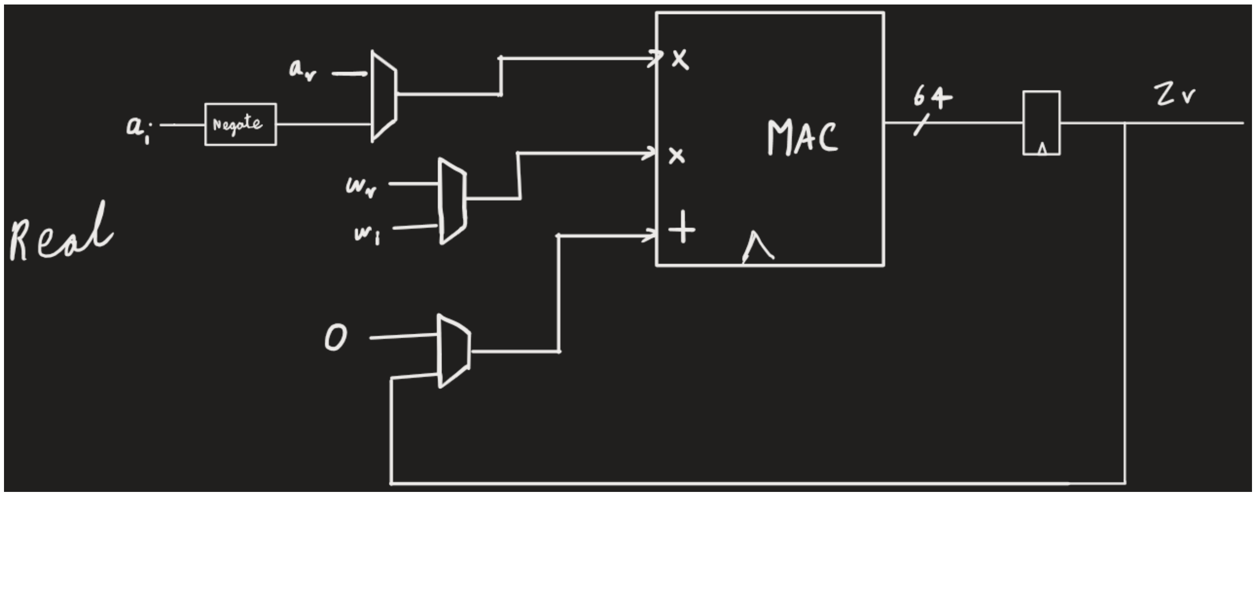
1. Input SRAM – Contains the input matrix along with number of qubits and gate matrices
2. Gates SRAM – Contains the gate/operator matrices
3. Output SRAM – Contains the final result
4. Scratchpad SRAM – A temporary data store used to store intermediate results

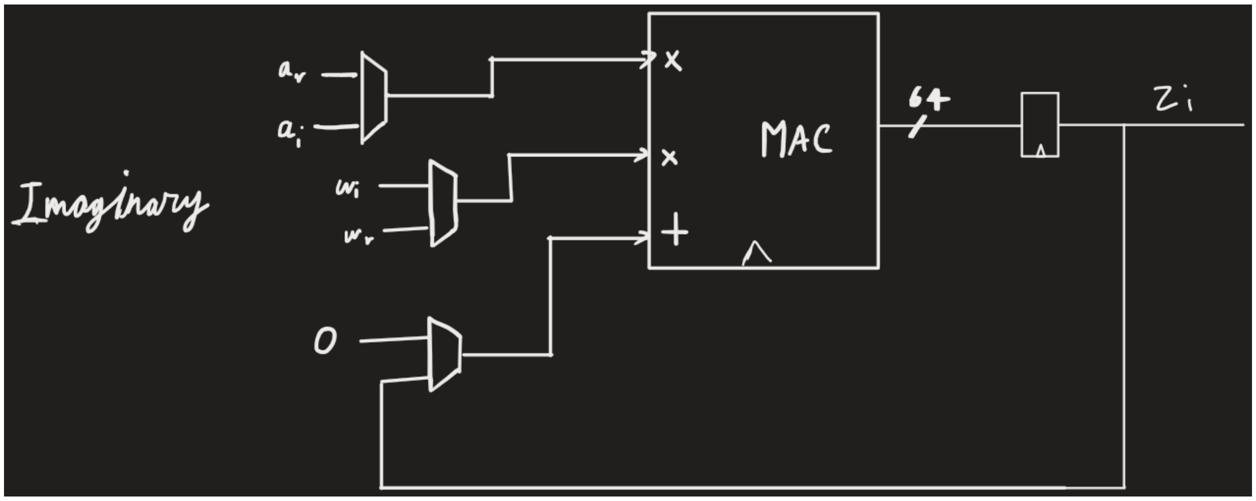
The block diagram below describes the high-level data flow:



For the first matrix multiplication, elements from the input SRAM and the gates SRAM are fetched and the result is stored in the scratchpad SRAM. For further matrix multiplications except the last, inputs to the MAC are fetched from scratchpad SRAM and gates SRAM and the results are fed back into the scratchpad SRAM. The only difference in operation of the last multiplication is that the result from the MAC is stored in the output SRAM.

The operational diagram of the 2 MAC units is shown below:





One MAC unit is used to compute the real number in the result, and another is used to calculate the imaginary number.

The intermediate results after one clock cycle are
Real MAC:
$(-a_i \times w_i) + 0 = -a_i \times w_i$

Imaginary MAC:
$(a_r \times w_i) + 0 = a_r \times w_i$

The final results after 2 clock cycles are
Real MAC:
$(a_r \times w_r) + (-a_i \times w_i) = (a_r \times w_r) - (a_i \times w_i)$

Imaginary MAC:
$(a_i \times w_r) + (a_r \times w_i) = (a_i \times w_r) + (a_r \times w_i)$

Final result = $[(a_r \times w_r) - (a_i \times w_i)] + i\,[(a_i \times w_r) + (a_r \times w_i)]$

The same came be extended to perform a full matrix multiplication, which would have values as below. The example uses a 2-qubit system with gate matrix (a, b, c, d…) and input matrix (w, x, y, z). Each element multiplication (computation of aw) requires 4 multiplications and 2 additions over 2 clock cycles. Computation of each row element would require 16 (4 x 4) multiplications and 11 ((4 x 2) + 3) additions.

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix} \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} aw + bx + cy + dz \\ ew + fx + gy + hz \\ iw + jx + ky + lz \\ mw + nx + oy + pz \end{bmatrix}$$

$$aw = (a_r w_r - a_i w_i) + i(a_i w_r + a_r w_i)$$

$$bx = (b_r x_r - b_i x_i) + i(b_i x_r + b_r x_i)$$

$$\vdots$$

3. Interface Specification

The design interfaces with the 4 above-mentioned SRAMs and the top module. The pins are as follows,

Pin Name	Width	Direction	Function
reset_n	1	Input	Reset signal
clk	1	Input	Clock
dut_valid	1	Input	Go signal (sent by top)
dut_ready	1	Output	Set high when design is idle
q_state_input_sram_write_enable	1	Output	Input SRAM write enable
q_state_input_sram_write_address	Q_STATE_INPUT_SRAM_ADDRESS_UPPER_BOUND	Output	Input SRAM write address

q_state_input_sram_write_data	Q_STATE_INPUT_SRAM_DATA_UPPER_BOUND	Output	Input SRAM write data
q_state_input_sram_read_address	Q_STATE_INPUT_SRAM_ADDRESS_UPPER_BOUND	Output	Input SRAM read address
q_state_input_sram_read_data	Q_STATE_INPUT_SRAM_DATA_UPPER_BOUND	Input	Input SRAM read data
q_state_output_sram_write_enable	1	Output	Output SRAM write enable
q_state_output_sram_write_address	Q_STATE_OUTPUT_SRAM_ADDRESS_UPPER_BOUND	Output	Output SRAM write address
q_state_output_sram_write_data	Q_STATE_OUTPUT_SRAM_DATA_UPPER_BOUND	Output	Output SRAM write data
q_state_output_sram_read_address	Q_STATE_OUTPUT_SRAM_ADDRESS_UPPER_BOUND	Output	Output SRAM read address
q_state_output_sram_read_data	Q_STATE_OUTPUT_SRAM_DATA_UPPER_BOUND	Input	Output SRAM read data
scratchpad_sram_write_enable	1	Output	Scratchpad SRAM write enable
scratchpad_sram_write_address	SCRATCHPAD_SRAM_ADDRESS_UPPER_BOUND	Output	Scratchpad SRAM write address
scratchpad_sram_write_data	SCRATCHPAD_SRAM_DATA_UPPER_BOUND	Output	Scratchpad SRAM write data
scratchpad_sram_read_address	SCRATCHPAD_SRAM_ADDRESS_UPPER_BOUND	Output	Scratchpad SRAM read address

scratchpad_sram_read_data	SCRATCHPAD_SRAM_DATA_UPPER_BOUND	Input	Scratchpad SRAM read data
q_gates_sram_write_enable	1	Output	Gates SRAM write enable
q_gates_sram_write_address	Q_GATES_SRAM_ADDRESS_UPPER_BOUND	Output	Gates SRAM write address
q_gates_sram_write_data	Q_GATES_SRAM_DATA_UPPER_BOUND	Output	Gates SRAM write data
q_gates_sram_read_address	Q_GATES_SRAM_ADDRESS_UPPER_BOUND	Output	Gates SRAM read address
q_gates_sram_read_data	Q_GATES_SRAM_DATA_UPPER_BOUND	Input	Gates SRAM read data

The parameters are tabularized below,

Parameter	**Value(s)**	**Description**
Q_STATE_INPUT_SRAM_ADDRESS_UPPER_BOUND	32	Input SRAM address width
Q_STATE_INPUT_SRAM_DATA_UPPER_BOUND	128	Input SRAM data width
Q_STATE_OUTPUT_SRAM_ADDRESS_UPPER_BOUND	32	Output SRAM address width
Q_STATE_OUTPUT_SRAM_DATA_UPPER_BOUND	128	Output SRAM data width
SCRATCHPAD_SRAM_ADDRESS_UPPER_BOUND	32	Scratchpad SRAM address width
SCRATCHPAD_SRAM_DATA_UPPER_BOUND	128	Scratchpad SRAM data width
Q_GATES_SRAM_ADDRESS_UPPER_BOUND	32	Gates SRAM address width
Q_GATES_SRAM_DATA_UPPER_BOUND	128	Gates SRAM data width

Assume a 2 Qubit with 4 operator matrices system for the examples. Write addresses are marked with **W**. Read addresses are not marked.

Loop through input SRAM to get the input matrix elements and iterate through gates SRAM to get operator matrix elements for the first matrix multiplication. Deploy 2 MAC units – one for calculating the real part of the result and another to calculate the imaginary part. Store the results in scratchpad SRAM linearly. Example – input and gates SRAM read address values and scratchpad SRAM write address values would be

i/p	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
gates	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
scrpd **W**				0				1				2				3

For the next matrix multiplications, switch to the scratchpad SRAM for state matrices. Continue iterating through the operator matrix to fetch gate matrices. Example – Example – scratchpad and gates SRAM read address values, and scratchpad SRAM write address values would be

scrpd	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
gates	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
scrpd **W**				4				5				6				7

scrpd	4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
gates	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
scrpd **W**				8				9				10				11

For the final matrix multiplication, switch to the output SRAM to store results. The following are the addresses.

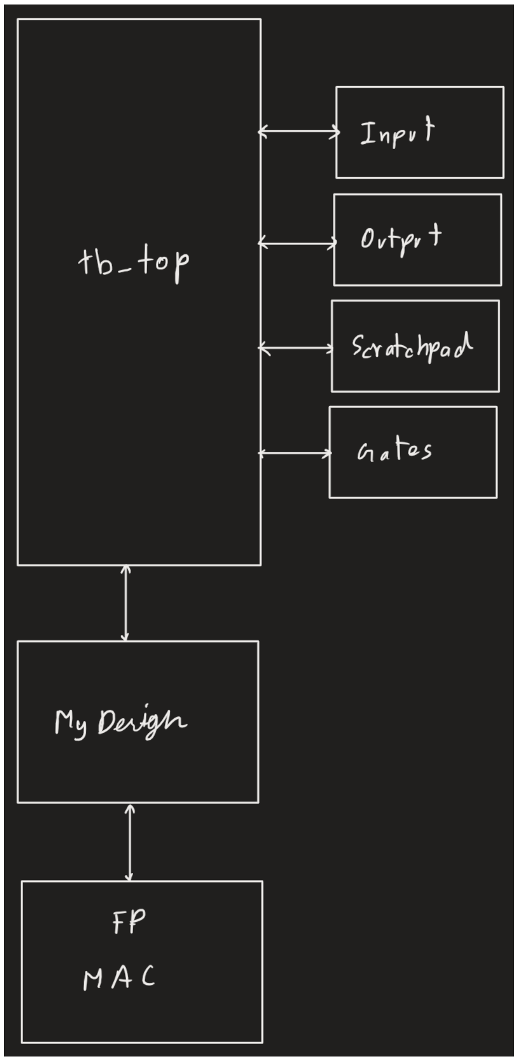
scrpd	8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
gates	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
o/p **W**				0				1				2				3

Within a matrix multiplication, each multiplication takes 2 clock cycles. So read address values would change every 2 cycles.

4. Technical Implementation

The design implements a 5 state Mealy machine to generate control signals. The first state is the idle state, and the next two states are used to cause delay until initial values are fetched from the SRAMs. The final two states are where matrix multiplication is performed, addresses are incremented, and outputs are written into the output lines.

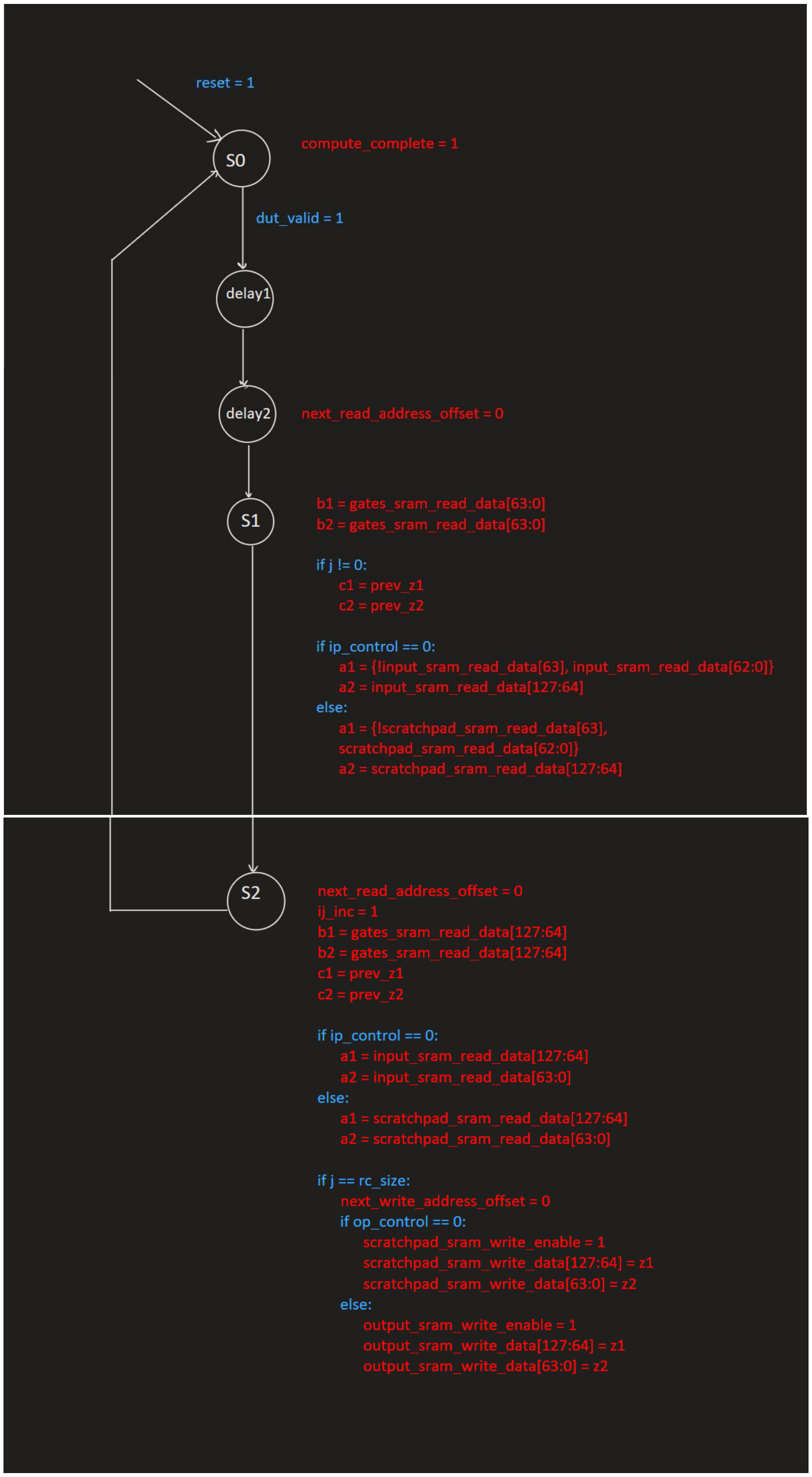
The block diagram shows the hierarchy of the various modules.



Below is the finite state diagrm of the control signals.
The values in blue are the inputs and the ones in red are the outputs. Functions of some of the pins:
a and b - multiplicative inputs to the MAC
c – Additive input to the MAC
i – row counter
j – colomn counter
ij_inc – rown and colomn increment value
next_read_address_offset – input and scratchpad SRAM read address increment value
next_write_address_offset – scratchpad and output SRAM write address increment value
ip_control – switch between input and scratchpad SRAM for reading inputs
op_control – switch between scratchpad and output SRAM for writing results
rc_size – matrix row size (equal to colomn size)

Default values:
compute_complete = 0
c1 = 0
c2 = 0
a1 = 0
a2 = 0
b1 = 0
b2 = 0

next_read_address_offset = 1
ij_inc = 0
next_write_address_offset = 0
output_sram_write_enable = 0
output_sram_write_data = 0
scratchpad_sram_write_enable = 0
scratchpad_sram_write_data = 0



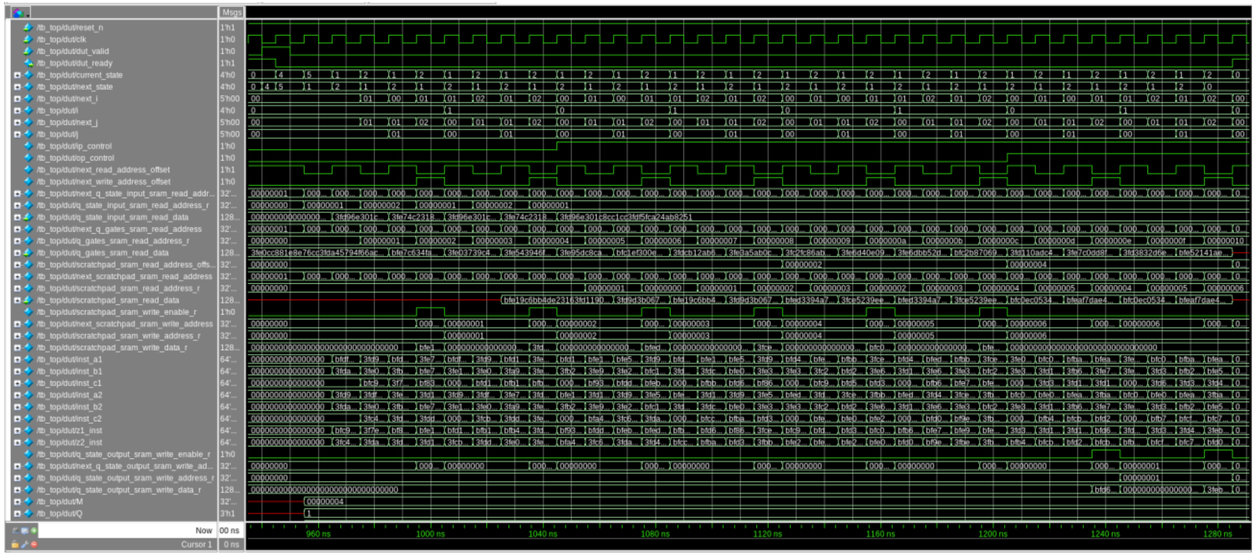
```
b1 = gates_sram_read_data[63:0]
b2 = gates_sram_read_data[63:0]

if j != 0:
    c1 = prev_z1
    c2 = prev_z2

if ip_control == 0:
    a1 = {!input_sram_read_data[63], input_sram_read_data[62:0]}
    a2 = input_sram_read_data[127:64]
else:
    a1 = {!scratchpad_sram_read_data[63],
    scratchpad_sram_read_data[62:0]}
    a2 = scratchpad_sram_read_data[127:64]
```

```
next_read_address_offset = 0
ij_inc = 1
b1 = gates_sram_read_data[127:64]
b2 = gates_sram_read_data[127:64]
c1 = prev_z1
c2 = prev_z2

if ip_control == 0:
    a1 = input_sram_read_data[127:64]
    a2 = input_sram_read_data[63:0]
else:
    a1 = scratchpad_sram_read_data[127:64]
    a2 = scratchpad_sram_read_data[63:0]

if j == rc_size:
    next_write_address_offset = 0
    if op_control == 0:
        scratchpad_sram_write_enable = 1
        scratchpad_sram_write_data[127:64] = z1
        scratchpad_sram_write_data[63:0] = z2
    else:
        output_sram_write_enable = 1
        output_sram_write_data[127:64] = z1
        output_sram_write_data[63:0] = z2
```

5. Verification

Test cases were generated using a python script covering all 4 qubit sizes and varying number of operator matrices. A total of 39 test cases were generated. The tb_top module can fetch the test cases and verify if the outputs are as expected. The outputs of the tb_top module include a summary of running the test cases, time duration and the number of clock cycles consumed.

A timing diagram of the design is shown below for the emulation of a 1 Qubit quantum computer and 4 gate matrices.



6. Results Achieved

The results of the simulation and synthesis are presented below:

Area achieved = 47574.8 um2
Clock period achieved = 40 ns
Clock frequency = 25 MHz
Number of clock cycles required to run Test 6 = 2572

7. Conclusion

A quantum computer for qubit sizes between 1 and 4 was designed using Verilog, simulated, and synthesized successfully. The design achieves a clock period of 40ns and utilizes an area of 47574.8 um^2.