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

Name: Nick Urch
Unityid: nurch
StudentID:. [redacted]

Delay (ns to run provided provided example).
Clock period: 20ns
cycles": 4096
20ns * 4096 = 81,920ns = 81.92µs

Logic Area: (um^2)

30202.9702

Memory: N/A

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

$$\frac{1}{81920ns * 30202.9702} = 4.04 \text{ e-10}$$

Delay (TA provided example. TA to complete)

1/(delay.area) (TA)

ECE464 Final Project Report – Quantum Emulator

Nick Urch

Abstract

The ECE 464 final project emulates a quantum computer with two Q-bits. The final design uses 64-bit IEEE floating point digits to multiply and accumulate multiple matrices and write the result to a static random-access memory (SRAM) unit. The device is simulated in Verilog using ModelSim and synthesized in Synopsys Design Vision.

1. Introduction

The next breakthrough in computer engineering uses quantum computers to greatly reduce computation times, improve on classical algorithms, and introduce new algorithms not possible with ordinary computers. Classic computers use bits comprising of either a one or zero. Quantum computers use Qubits, which can be in a superposition of one ***and*** zero until the state of the bit is measured. Although quantum computing has been around for nearly a decade, the production costs in time and money make an emulator desirable for testing and pre-production.

2. Micro-Architecture

The high-level architecture comprises four SRAM units that can be read or written from along with a floating point multiplier-accumulator (FP MAC). The Q_input SRAM includes a 4x1 matrix "A" that will be multiplied with the Q_gate SRAM 4x4 matrix "B". There could be multiple B matrices that would have to be multiplied with the previous output. The final 4x1 matrix is then written to the Q_output SRAM.

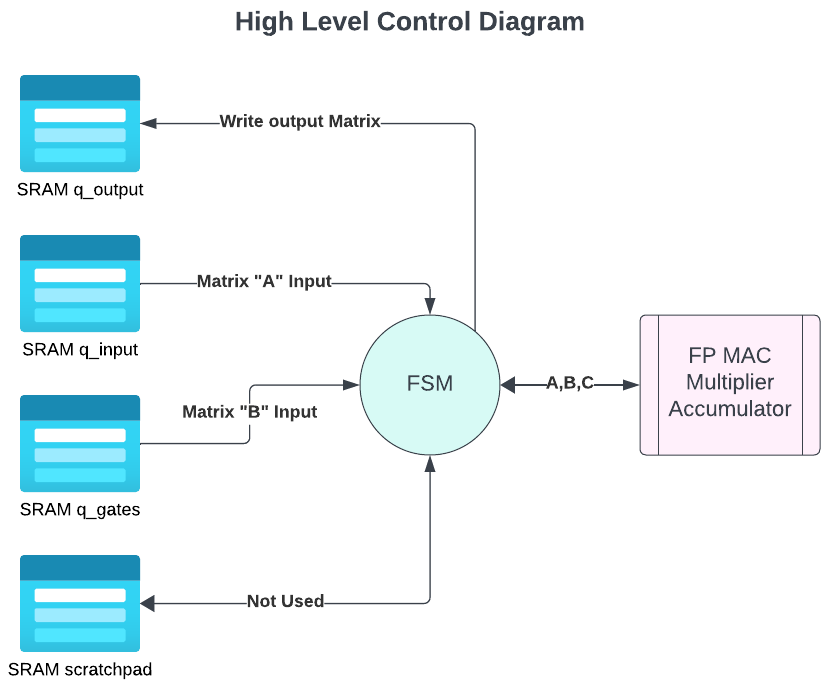

Figure 1 – High-level architecture

3. Interface Specification

The top-level module is controlled by a finite state machine (FSM) with eight discrete states from zero to seven. State 0 is the reset/waiting state. When the signal dut_valid is received, the FSM transitions to State 1. At State 1, the q_input data is read from the SRAM starting with address zero. State 1 will then advance the q_input read address, and the state transitions to State 2. State 2 is a waiting cycle for the SRAM to read the new address. If the address is greater than four, the state will transition to State 3. State 3 reads the data from the q_gate SRAM and then moves to State 4. State 4 pushes the "A" and "B" matrix rows/columns to be multiplied and accumulated into the FP MAC and then transitions to State 5. During State 5, the data from the FP MAC is saved to the outMatrix register, and the following q_gate address is applied. If there are still more computations to be implemented, the state moves back to State 3 to repeat or moves to State 6 if the calculation is complete. On State 6, the write data and address are written to the q_output registers along with the write enable, and the state moves to State 7. State 7 is a write wait cycle that allows the data to be written into q_output SRAM. If all four entries are written, compute_complete is written high, telling the testbench the output is ready and moves to State 0 to wait, or else advances the output address and returns to State 6. The state diagram is shown below in Figure 2.

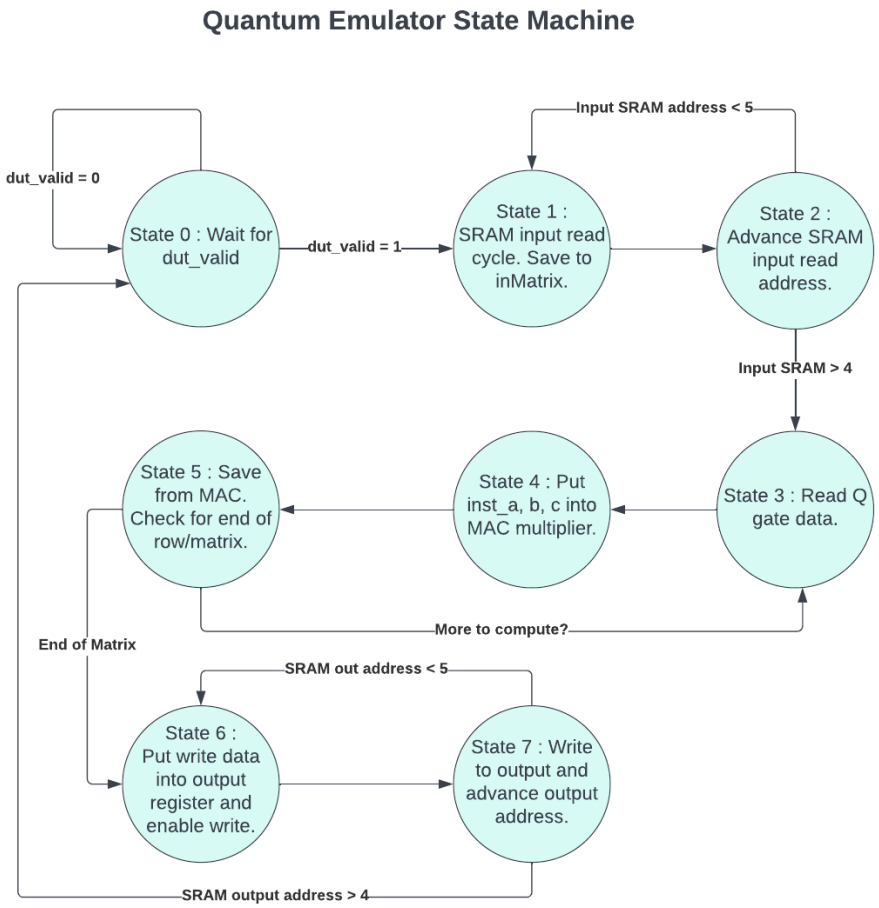

Figure 2 – FSM Diagram

Along with the FSM, multiple memory devices are needed to keep track of current variables. All memory devices used in this application are D flip-flops with synchronous reset. The name, bit length, and details are shown below in Figure 3.

Memory Devices			
Name	**Type**	**Width (Bits)**	**Details**
curState	Flip-flop	3	State Machine Register
compute_complete	Flip-flop	1	Tells test bench when finished
numMatrix	Flip-flop	8	Total Q Gate matrices
curMatrix	Flip-flop	8	Current Matrix being computed
q_gates_sram_read_address_r	Flip-flop	32	SRAM read register
q_state_input_sram_read_address_r	Flip-flop	32	SRAM read register
inMatrix	Flip-flop	256	Input Matrix (4x64 bit)
outMatrix	Flip-flop	256	Output Matrix from MAC (4x64 bit)
curRowA	Flip-flop	2	Current row of Matrix "A" being multiplied
curRowB	Flip-flop	2	Current row of Matrix "B" being multiplied
q_state_output_sram_write_data_r	Flip-flop	128	Data to be written to SRAM
q_state_output_sram_write_enable_r	Flip-flop	1	SRAM write enable
inst_a	Flip-flop	64	MAC instance A to multiply
inst_b	Flip-flop	64	MAC instance B to multiply
inst_c	Flip-flop	64	MAC instance C to accumulate
inst_rnd	Flip-flop	3	MAC rounding register
scratchpad_sram_write_address_r	Flip-flop	32	Scratch SRAM Write register
scratchpad_sram_write_data_r	Flip-flop	128	Scratch SRAM write data
scratchpad_sram_write_enable_r	Flip-flop	1	Scratch write enable register
q_state_input_sram_write_address_r	Flip-flop	32	SRAM write address register
q_state_input_sram_write_data_r	Flip-flop	128	SRAM write data
q_state_input_sram_write_enable_r	Flip-flop	1	SRAM write enable
q_state_output_sram_read_address_r	Flip-flop	32	SRAM read register
q_gates_sram_write_address_r	Flip-flop	32	SRAM write address register
q_gates_sram_write_data_r	Flip-flop	128	SRAM data register
q_gates_sram_write_enable_r	Flip-flop	1	SRAM write enable
scrachpad_sram_read_address_r	Flip-flop	32	SRAM read register

Figure 3 – Memory Devices

4. Technical Implementation

Implementation of the matrix multiplier is achieved by a 2-bit curRowB variable that will advance each time with the q_gate address. Since all matrices are 4x4, the curRowB variable will overflow back to 00 to start the new row. The FSM will check for the end of the current matrix by using curMatrix variable shifted by 4 (if q_gate_read_address_r = curMatrix << 4). If the end of the matrix is found, the outMatrix is fed into the inMatrix and the next matrix is calculated. The final value of outMatrix is then written to the q_output SRAM and compute_complete is set high to tell the testbench the value is written. A sample waveform of the computation of a row and column in the FP MAC is shown below in Figure 4.

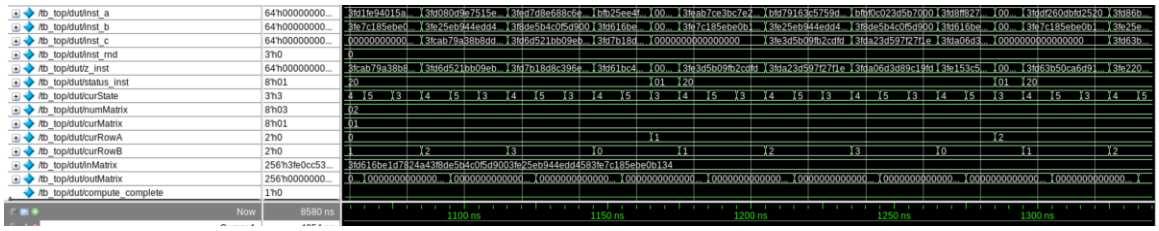

Figure 4 – Waveform of computation.

5. Verification

Verification and debugging were achieved through the ModelSim wave display. The Verilog code was written using an iteration approach. State 0 was verified to work, then State 1, then State 2, etc… When the satisfactory results were met, “make eval-464” was typed to the MobaXterm console, and the results are shown below in Figure 5.

```
#
# INFO: Total number of cases  : 32
# INFO: Total number of passes : 32
# INFO: Finial Results         : 100.00
# INFO: Finial Time Result     : 20480 ns
# INFO: Finial Cycle Result    : 4096 cycles
#
# ** Note: $finish    : ../testbench/testbench.sv(348)
#    Time: 21220 ns  Iteration: 1  Instance: /tb_top
# End time: 16:25:39 on Nov 27,2023, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

Figure 5 – Test Evaluation.

6. Results Achieved

The results of the emulator are a 20ns clock speed (50 Mhz) and all tests were completed in 4096 cycles. The final surface area of the synthesized chip is 21,305 total cells with an area of 30,202.9702 µm.

7. Conclusions

Quantum computing is the future of microarchitecture, but until these applications are cost-effective for the average user, emulators will need to be created to reduce the cost and time needed to build and test these complex machines. This emulator displays the capabilities of these incredible machines.