200283294. Name: Ibrahim Moghul		
Delay 33054 ns . Clock period: 21 # cycles”: 1574	Area: 79750.7905 (um^2) Logic: 76188.518 Memory: N/A	3.7935078e-10 ($ns^{-1}.um^{-2}$)
Delay (TA provided example. TA to complete)		1/(delay.area) (TA)

Abstract

With the rise of research in the area of quantum computing, many people are starting to get involved in learning quantum computing basics. Since quantum computers are not exactly portable, many people are forced to resort to remotely running their programs on a remote service that allows them a turn. The goal of the project is to allow the widespread simulation of quantum circuits for virtually anyone by way of a quantum computing simulation accelerator. By offloading a quantum simulation to an ASIC, sunning these simulations becomes more accessible and quicker, as network latency is no longer a problem. Though it won’t run as fast as it would on a quantum computer, this ASIC is a very good alternative to waiting for a simulation to run remotely.

ASIC Design of Quantum Computing Simulation Accelerator

Ibrahim Moghul

Abstract

With the rise of research in the area of quantum computing, many people are starting to get involved in learning quantum computing basics. Since quantum computers are not exactly portable, many people are forced to resort to remotely running their programs on a remote service that allows them a turn. The goal of the project is to allow the widespread simulation of quantum circuits for virtually anyone by way of a quantum computing simulation accelerator. By offloading a quantum simulation to an ASIC, sunning these simulations becomes more accessible and quicker, as network latency is no longer a problem. Though it won't run as fast as it would on a quantum computer, this ASIC is a very good alternative to waiting for a simulation to run remotely.

Report

This design makes use of and interfaces with 4 SRAMs. The first holds the quantum gates, the second holds the input state, the third takes the output state and the fourth is a scratchpad used for offloading the intermediate states. The data in these SRAMs are all complex 64 bit floating point numbers. The floating point arithmetic is done using Design Ware IP, with a fully pipelined arithmetic unit that produces results every cycle. The design is very fast and the algorithm is O(N) meaning that it will complete the simulation in nearly the same amount of cycles as there are data entries. With this high throughput, area tradeoffs, and low clock period, this design minimizes PPA considerations. This report will explain the implementation/micro architecture, specifications, verifications, and some of the results achieved.

My design takes advantage of the scratchpad. The product of the first operator matrix(in q_gates SRAM) and the initial input states(in q_input_states SRAM) is calculated by iteration row by row and placed in the scratchpad SRAM as well as the q_output_states SRAM. As described earlier, the floating point arithmetic unit will produce a result every cycle. When that is finished, the matrix in the scratchpad will be read along with the next operator matrix and the product will be places in q_input_states and q_output_states. This toggle will continue until completion. With this algorithm, during the computation for each operator matrix, every SRAM is either read from or written to. This allows the FPU to be fed new values on every cycle. All in all, this substantially reduces the number of cycles required.

Signals, width and function (keep in my there are 4 different srams, meaning the last 5 entries will be expanded to 20 different entries):

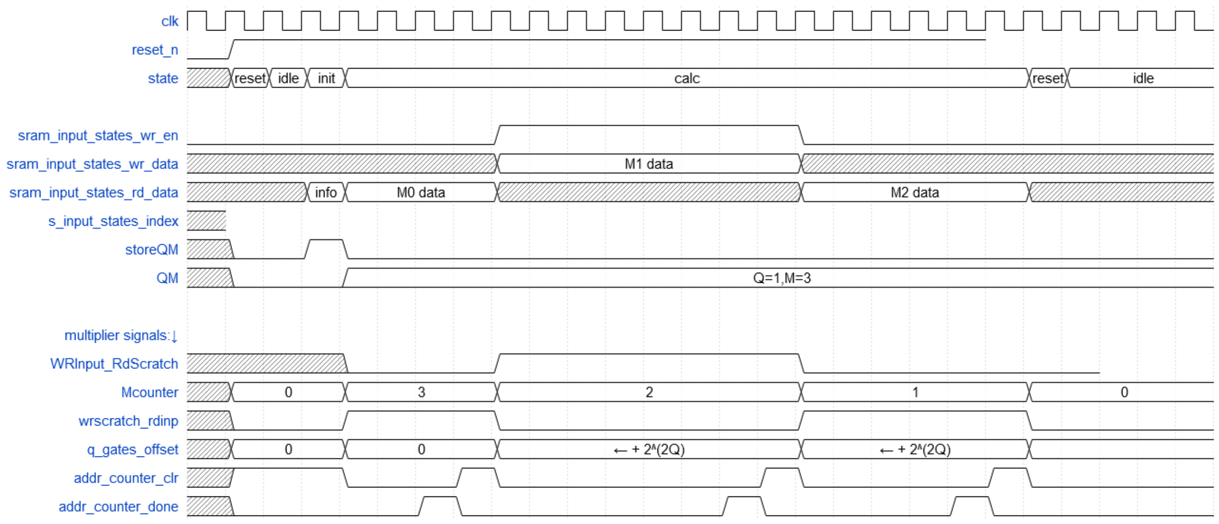
Signal	Direction	Width	Function
Reset_n	input	1	Active low reset
Clk	input	1	Clock signal
Dut_valid	input	1	Drive this high to start calculation
Dut_ready	output	1	Signals whether the system is ready or not
<sram_name>_write_enable	output	1	Active high enable for write, low for read
<sram_name>_write_address	output	32	The address to write to

<sram_name>_write_data	output	128	The data to write
<sram_name>_read_address	output	32	The address to read from
<sram_name>_read_data	input	128	The data that is read

This design has certain assumptions it makes:

1. The 1st element in the q_inputs_states SRAM contains the concatenation of Q and M which are the number of qubits and number of operator matrices. Each takes half the number of bits that a location in the SRAM takes.
2. Qubits will be from 1 to 4.
3. Number of operator matrices will not exceed 20.
4. The dut_valid signal during operation (when dut_ready is low) is ignored.

A timing diagram of the operation is shown below:



To optimize area, all the address counters for the 4 SRAMS were controlled by 1 main counter. Bitmasks were used to obtain different parts of that counter, therefore decreeing asing the number of adders required. When determining how to chunk up this counter, using the number of qubits is required. There were three options to perform a shift, which is required to find the dimensions of the matrix. The first is a normal shift register. This is a viable option but increases the cycle count. The next is a barrel shifter, but the size of this is not worth it. I resorted to using a truth table, as the max number of qubits is known. This allows logic optimizations to be made as desired using don’t care values.

Verification was achieved by using pre-simulated runs, and checking the final output in the output sram, with the simulated values. Verification was also performed for features such as reset and bring-up. This was all done by a testbench which ran different simulations, without resetting. This tested the state machine’s ability to properly reset after completing a simulation.

The design was optimized for Area, clock period, and cycle count. Some of the considerations were mentioned in earlier sections. With the decisions made, this design was high on area but extremely effective in throughput, while achieving modest energy consumption. The area was around 78000 um^2. The clock period achieved as 24 ns and was limited to that by the floating-point multiplier. A total of 1574 cycles were needed to run 4 simulations. The first one having 1 qubit and 4 operator matrices. The second having 2 qubits and 5 operator matrices. The third having 1 qubit and 20 operator matrices. And the fourth having 4 qubits and 4 operator matrices. All in all, the trade off for area makes sense after looking at the throughput results.

To conclude, this design helps accelerate quantum simulations, by offloading it to an ASIC. The design, takes in the configuration and values from the SRAMs, and outputs it to one of them. Though the goal was to optimize for area, clock period, and cycle count but it seems the most well balanced design prioritized throughput which is what this design does.