Project Plan report format – Your first page MUST match this format /10

Name: Yazhuo Gao
Unityid: ygao46
StudentID: 200542955

Summary Risk Plan:

Verification plan:

The main parts to be verified are data input and multiplication.
To verify data input, display the data on terminal as well as check if they are stored in matrices correctly.
To verify multiplication, display results and compare them with hand-computation, and also be sure that the results are stored correctly.

Risk:

1. The multiplication may cause overflow of bits, which results in the missing or mismatch of results.
2. The computation may have weird operation with imaginary part.
3. Unsure of how to optimize performance by freeing the memory.

Schedule:
Complete dut draft (11/15)
Debug (11/17)
Optimize (11/22)

Brief Description of Mode of operation, including selected algorithms:

- Use reset_n to reset the dut
- Set dut_ready to 0 after dut_valid is 1
- Begin reading data after dut_ready is 0
- Use combination of module DW_fp_mult and DW_fp_mac to compute result
- Write data after computation
- Set dut_ready back to 1

High level sketch. Add details on the following pages if necessary

Design Blocks:

DUT&SRAM {
    Qubits {
        input q_state_input
        output q_state_output
    }
    Input q_gates
    Module multiplier
}

Special cases:
    Real and real
    Real and mixed imaginary
    Imaginary and imaginary

Logic:
$Conditional\ NOT \times Hadamard\ operator \times Qubits = Output$
Conditional NOT is fixed
Receive input data of Hadamard operator and Qubits separately

Main part: the multiplier
Since mixed imaginary numbers are included, we need separate multipliers for real and imaginary numbers.

Module multiplier {

- Construct six two-dimension variables as the matrices: operator, qubits, result, and three distinct variables that store the imaginary part of them.

- Firstly, deal with the multiplication of conditional NOT and the operator: conditional NOT contains real numbers while the operator might contain real or mixed imaginary numbers.
- If the operator has real numbers, use module DW_fp_mult to multiply the two matrices. (real and real)
- The actual operation would function like the c++ code below:

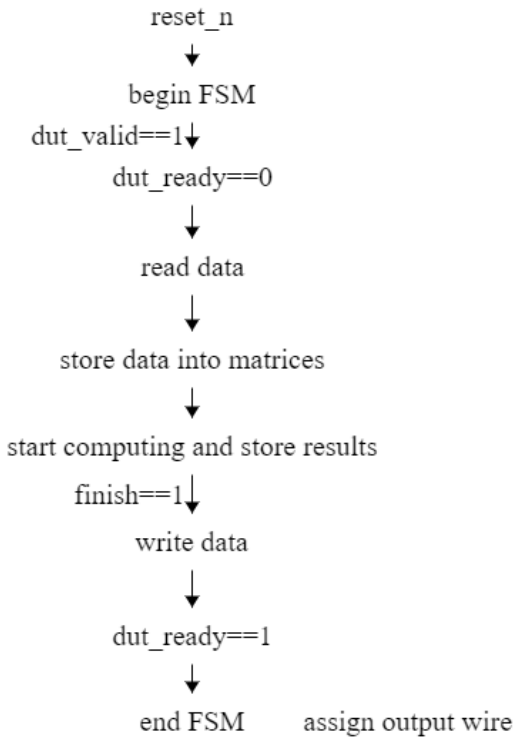
```
for(i = 0; i < r1; ++i)
        for(j = 0; j < c2; ++j)
                for(k = 0; k < c1; ++k)
                {
                        mult[i][j] = a[i][k] * b[k][j];
                }
```

- The input value for multiplication will be selected from rows and columns of matrices separately.
- If the operator has mixed imaginary numbers, use module DW_fp_mult to compute the multiplication of conditional NOT and the imaginary part of the operator, then use module DW_fp_mac to multiply the real part of the matrices and add the result of imaginary part. (real and imaginary)

- Then deal with the multiplication with qubits: the multiplication can be the three special cases described above; since we have dealt with real&real and real&imaginary in previous line, now we leave imaginary&imaginary to be done.
- If both matrices have mixed imaginary numbers, use module DW_fp_mult to compute the multiplication of the imaginary part of both matrices, then use module DW_fp_mac to multiply the real part of the matrices and add the result of imaginary part. (imaginary and imaginary)

}

Use a Finite State Machine to deal with the input data, computation, and output result.



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

Name: Yazhuo Gao
Unityid: ygao46
StudentID:. 200542955

Delay (ns to run provided provided example).
Clock period: 5ns
cycles": 6204

Logic Area: (um^2)

Memory: N/A

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

Delay (TA provided example. TA to complete)

1/(delay.area) (TA)

Quantum Computing Emulator

Yazhuo Gao

Abstract

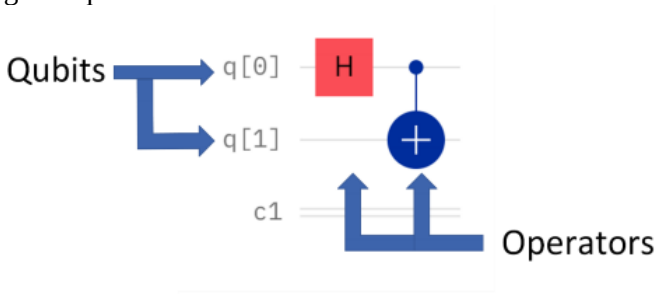
This program is a Quantum Computing Emulator that emulates the basic computation logic of quantum computing.

1. Introduction

- Design Description

The quantum computing emulator loads data from two separate files, which represent the qubits value and the gate (Hadamard operator) value, and compute the result by multiplying matrices constructed by these values through the logic of quantum computing. The results describe the probabilistic state of the system.

Logic of quantum circuit:

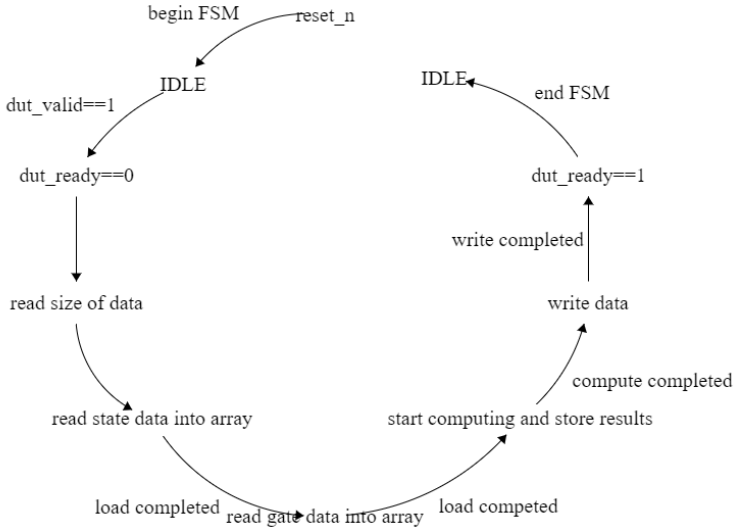


- Key innovations

The design interacts with SRAM and Top module to perform basic functionalities and testing. Module DW_fp_mac (Floating-Point Multiply-and-Add) is also instantiated in the design as part of the logic operation which multiplies two matrices. Therefore, the design can accept floating point values as input and compute results precisely.

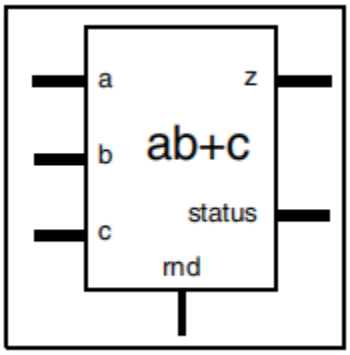
2. Micro-Architecture

- Finite State Machine



The program uses Finite State Machine as a loop, including eight states, to achieve the overall work. The FSM allows to complete each work step by step, which reduces potential bugs as well as ensures stability.

- DW_fp_mac



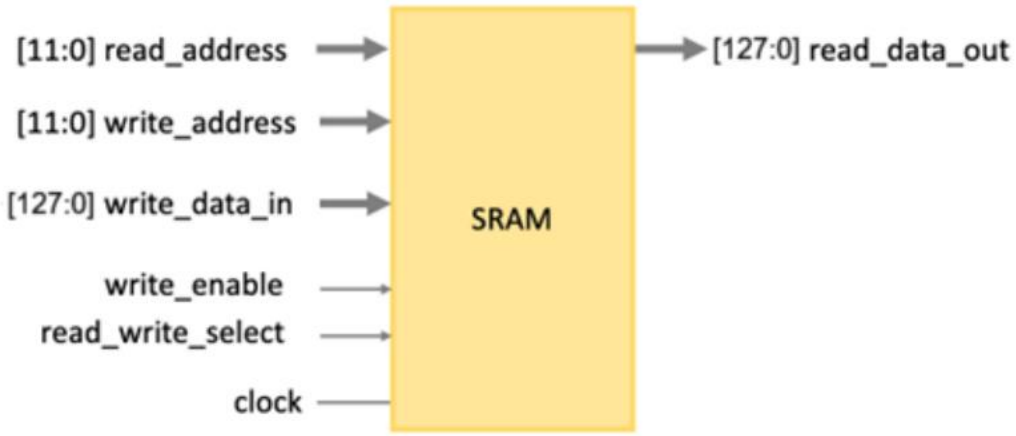
DW_fp_mac is a floating-point component that performs the multiply and add operation. It sums up a floating-point product of input a and b to input c (ab + c) to produce a floating point multiply and add result, z.

3. Interface Specification

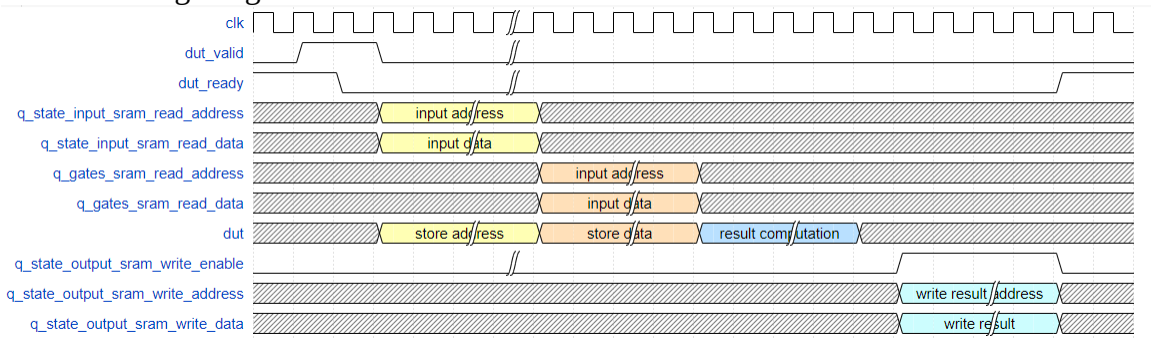
- Detailed description of top level interface

Top level interface		
Signal	**Width**	**Description**
clk	**1**	**General clock that interacts with SRAM and Dut**
reset_n	**1**	**Reset modules if it's low**
dut_valid	**1**	**SRAM tells Dut to receive a new stack of data if dut_valid is asserted**
dut_ready	**1**	**Dut holds dut_ready low during FSM running, and sets high after the data is populated to SRAM**
q_state_output_sram_write_enable	**1**	**The conditional signal that tells dut to write data into SRAM**
q_state_output_sram_write_address	**32**	**The interface signal that write address from dut to SRAM**
q_state_output_sram_write_data	**128**	**The interface signal that write data from dut to SRAM, with real number at [127:64] and imaginary number at [63:0]**
q_state_input_sram_read_address	**32**	**The interface signal that read state data address from SRAM to Dut**
q_state_input_sram_read_data	**128**	**The interface signal that read state data from SRAM to Dut, with real number at [127:64] and imaginary number at [63:0]**
q_gates_sram_read_address	**32**	**The interface signal that read gate data address from SRAM to Dut**
q_gates_sram_read_data	**128**	**The interface signal that read gate data from SRAM to Dut, with real number at [127:64] and imaginary number at [63:0]**

SRAM interface



- Timing Diagram



4. Technical Implementation

Two-dimension arrays are constructed to store data for input, computing, and output.
FSM allows DW_fp_mac to input and compute one pair of floating numbers every clock phase.
In this case, I split two instantiations of DW_fp_mac to compute real and imaginary values separately, and rejoin them at the end.

5. Verification

- Approach used to verify correctness.

Example files including two input files and one output result file are created to verify the correctness of the program.

6. Results Achieved

The program now can produce results at precision of 7 decimals.

7. Conclusions

- Summary of project and key results

The project models a simple quantum computing emulator that achieves basic logic of quantum computing. An interface of top, SRAM, and dut module is designed to interacts between them and achieve the work of input, computing, and output. The maximum allowed inputs are

Input	Q	M	Address	Data
Max allow	4	20	32 bits	128 bits

The output result can be precise at 7 decimals.