ECSE 325

Lab 2 Report

Lucas Bluethner, 260664905

Yutian Jing, 260680087

March 23, 2018

Introduction

The purpose of this lab was to learn the basics of fixed-point representations, VHDL testbench creation, and functional verification using ModelSim. Specifically, we implemented a multiplication-accumulation (MAC) unit using a fixed-point representation in VDHL, and wrote code for a testbench to validate our design in ModelSim.

Code Implementation

We were given two files (lab2-x.txt and lab2-y.txt) containing 1000 floating-point numbers each. The first task was to write a program to convert these floating-point numbers to fixed-point and write them to two new files (lab2-x-fixed-point.txt and lab2-y-fixed-point.txt). Below is the the C code we wrote to implement this conversion.

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

char *decimal_to_binary(int);

/*
 * This method reads the floating point values from to separate files,
 * convert them to fixed point, and stores them in two new files
 */
void main()
{
    // Open the files
    FILE *lab2_x = fopen("lab2-x.txt", "r");
    FILE *lab2_y = fopen("lab2-y.txt", "r");

    // Create arrarys to hold the x values and y values
    int N = 1000;
    float x_values[N];
    float y_values[N];
    int i;

    // Read each floating point number from the files and store them in their corresponding array
    for (i = 0; i < N; i++)
    {
        fscanf(lab2_x, "%f", &x_values[i]);
    }
    for (i = 0; i < N; i++)
    {
        fscanf(lab2_y, "%f", &y_values[i]);
    }

    // Done with these files, close them
    fclose(lab2_x);
    fclose(lab2_y);

    // Go through th array multiplying each number by 2^7 because
    // b = a x 2^F where a = floating point number, F = fractional length
    // after this each value in the array will be integer values (so there will be no truncation)
    for (i = 0; i < N; i++)
    {
        x_values[i] = x_values[i] * 128;
    }
    for (i = 0; i < N; i++)
    {
        y_values[i] = y_values[i] * 128;
    }

    // Open the files where the fixed point values will be written to
    FILE *lab2_x_fixed_point = fopen("lab2-x-fixed-point.txt", "w");
    FILE *lab2_y_fixed_point = fopen("lab2-y-fixed-point.txt", "w");

    int n;
    char *pointer;

    // iterate through each number, convert it to binary (2's complement) then print it to it's corresponding file
    for (i = 0; i < N; i++)
    {
        n = (int)x_values[i];
        pointer = decimal_to_binary(n);
        fprintf(lab2_x_fixed_point, "%s\n", pointer);
    }
    for (i = 0; i < N; i++)
    {
        n = (int)y_values[i];
        pointer = decimal_to_binary(n);
        fprintf(lab2_y_fixed_point, "%s\n", pointer);
    }

    // Free the pointer from memory and close the files because we are done with them
    free(pointer);
    fclose(lab2_x_fixed_point);
    fclose(lab2_y_fixed_point);
}

/*
 * This method take an integer as an argument an converts it to a 10-bit binary number
 * If the integer is negative, the number gets convert to its positive binary value,
 * then gets converted again to 2's complement
 * returns a pointer to the MSB of the binary number in memory
 */
char *decimal_to_binary(int n)
{
    int c, d, count, i, invert_spot;
    char *pointer;

    int neg_number = 0;

    // initialzer the counter
    count = 0;

    // will point to the MSB (sign bit) of the binary number in memory
    pointer = (char *)malloc(10 + 1);

    // Array to hold the binary digits
    int binary_digits[10];

    if (pointer == NULL)
        exit(EXIT_FAILURE);

    // If the number is negative, make it positive and set the neg_number flag high
    if (n < 0)
    {
        n = n * (-1);
        neg_number = 1;
    }

    // Converts the decimal number to its unsigned binary value storing it bit by bit in the array
    for (c = 9; c >= 0; c--)
    {
        d = n >> c;

        if (d & 1)
            binary_digits[count] = 1;
        else
            binary_digits[count] = 0;
        count++;
    }

    // if the argument n was postive, simply copy the values from the array to memory starting at the pointer
    if(!neg_number)
    {
        for(i = 0; i < 10; i++)
        {
            if(binary_digits[i] == 0)
            {
                *(pointer + i) = 0 + '0';
            }
            else
            {
                *(pointer + i) = 1 + '0';
            }
        }
    }
    else
    // the argument n was positive, convert to 2's complement
    {

        // Starting from the LSB, iterate through the array backwards until the first '1'
```

```
bit is reached
        // All the numbers before (and not including) this 1 value need to be inverted
        for (i = 9; i >= 0; i--)
        {
            if (binary_digits[i] == 1)
            {
                invert_spot = i;
                break;
            }
        }
        // Starting at the MSB, iterate throught the array storing the inverted value in memory starting at the pointer
        for (i = 0; i < 10; i++)
        {
            if (i < invert_spot)
            {
                if (binary_digits[i] == 0)
                {
                    *(pointer + i) = 1 + '0';
                }
                else
                {
                    *(pointer + i) = 0 + '0';
                }
            }
            else
            // Once the '1' found above is reached, simply copy the values over from the array to the next spot in memory
            {
                if (binary_digits[i] == 0)
                {
                    *(pointer + i) = 0 + '0';
                }
                else
                {
                    *(pointer + i) = 1 + '0';
                }
            }
        }
    }

    // Add the null terminator to the end of the string
    *(pointer + count) = '\0';

    // return a pointer the the MSB in memory
    return pointer;
}
```

Next, now that we had two files containing 1000 fixed-point numbers each, was to implements the MAC unit. The MAC unit takes in a number from each file once every cock cycle, multiplies them and accumulates to the current total.

Below is the VHDL code that implements the MAC unit.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity g64_lab2 is
port( x      : in std_logic_vector (9 downto 0); -- first input
        y        : in std_logic_vector (9 downto 0); -- second input
        N        : in std_logic_vector (9  downto 0); -- total number of inputs
        clk  : in std_logic; -- clock
        rst    : in std_logic; -- asynchronous active-high reset
        mac  : out std_logic_vector (19 downto 0); -- output of MAC unit
        ready  : out std_logic); -- denotes the validity of the mac signal
end g64_lab2;

architecture behaviour of g64_lab2 is

signal temp     : std_logic_vector(19 downto 0);
signal counter : std_logic_vector(9  downto 0);

begin
    process(clk)
    begin
        if (rst = '1') then -- asynchronous active-high reset
                temp <= (others => '0');     -- reset temp to zero
                counter <= (others => '0'); -- reset counter to zero
                ready <= '0';                        -- not ready
        elsif (rising_edge(clk)) then
            if (counter < N) then -- if counter is less than N (1000)
                    -- multiply two the two inputs, and accumulate
                    temp <= std_logic_vector(signed(x) * signed(y) + signed(temp));
                    -- increment the counter
                    counter <= std_logic_vector(signed(counter) + 1);
            end if;
        end if;
        ready <= '1'; -- ready
    end process;
    mac <= std_logic_vector(temp);
end behaviour;
```

Results

Testbench Code & Simulation

Below is the testbench code we wrote to verify the functionality of the MAC unit.

```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use STD.textio.all;
use ieee.std_logic_textio.all;

entity g64_MAC_tb is
end g64_MAC_tb;

architecture test of g64_MAC_tb is

-- Declare the Component Under Test
component g64_lab2 is
    port( x          : in std_logic_vector(9 downto 0); -- first input
          y          : in std_logic_vector(9 downto 0); -- second input
          N          : in std_logic_vector(9  downto 0); -- total number of inputs
          clk        : in std_logic; -- clock
          rst        : in std_logic; -- asynchronous active-high reset
          mac        : out std_logic_vector(19 downto 0); -- output of MAC unit
          ready      : out std_logic); -- denotes the validity of the mac signal
end component g64_lab2;

-- Testbench Internal Signals
file file_VECTORS_X : text;
file file_VECTORS_Y : text;
file file_RESULTS   : text;

constant clk_PERIOD : time := 100 ns;

signal x_in     : std_logic_vector(9 downto 0);
signal y_in     : std_logic_vector(9 downto 0);
signal N_in     : std_logic_vector(9  downto 0);
signal clk_in       : std_logic;
signal rst_in       : std_logic;
signal mac_out      : std_logic_vector(19 downto 0);
signal ready_out    : std_logic;

begin
    -- Instantiate MAC
    g64_MAC_INST    : g64_lab2
        port map (
            x => x_in,
            y => y_in,
            N => N_in,

            clk => clk_in,
            rst => rst_in,
            mac => mac_out,
            ready => ready_out
        );

    -- Clock generation
    clk_generation  : process
    begin
        clk_in <= '1';
        wait for clk_PERIOD/2;
        clk_in <= '0';
        wait for clk_PERIOD/2;
    end process clk_generation;

    -- Feeding Inputs
    feeding_instr   : process is
        variable v_Iline1   : line; -- will be one line read from one input file
        variable v_Iline2   : line; -- will be one line read from the other input file
        variable v_Oline    : line; -- variable to hold to output of the MAC
        variable v_x_in     : std_logic_vector(9 downto 0);
        variable v_y_in     : std_logic_vector(9 downto 0);
    begin
        -- reset the circuit
        N_in <= "1111101000"; -- N = 1000
        rst_in  <= '1';
        wait until rising_edge(clk_in);
        wait until rising_edge(clk_in);
        rst_in <= '0';
        file_open(file_VECTORS_X, "lab2-x-fixed-point.txt", read_mode);
        file_open(file_VECTORS_Y, "lab2-y-fixed-point.txt", read_mode);
        file_open(file_RESULTS, "lab2-out.txt", write_mode);

        while not endfile(file_VECTORS_X) loop
            readline(file_VECTORS_X, v_Iline1);
            read(v_Iline1, v_x_in); -- read one number from first input file
            readline(file_VECTORS_Y, v_Iline2);
            read(v_Iline2, v_y_in); -- read one number from second input file

            x_in <= v_x_in;
            y_in <= v_y_in;

            wait until rising_edge(clk_in);
        end loop;

        if ready_out = '1' then
            write(v_Oline, mac_out); -- write the result of the MAC to the ouput file
                                   -- link it to mac_out signal
            writeline(file_RESULTS, v_Oline);
            wait;
        end if;
    end process;
end architecture test;
```

Running the simulation, it was verified that the MAC unit functioned as expected, as seen in Fig. 1 below.

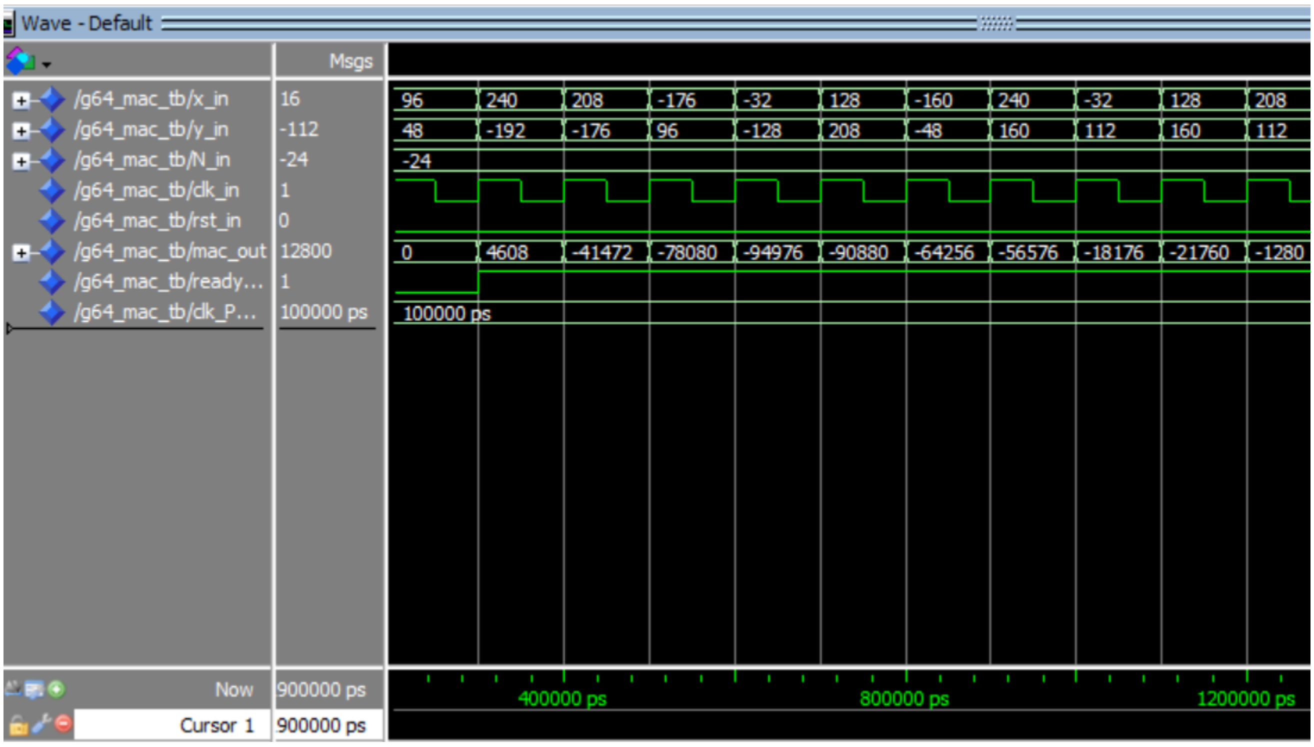
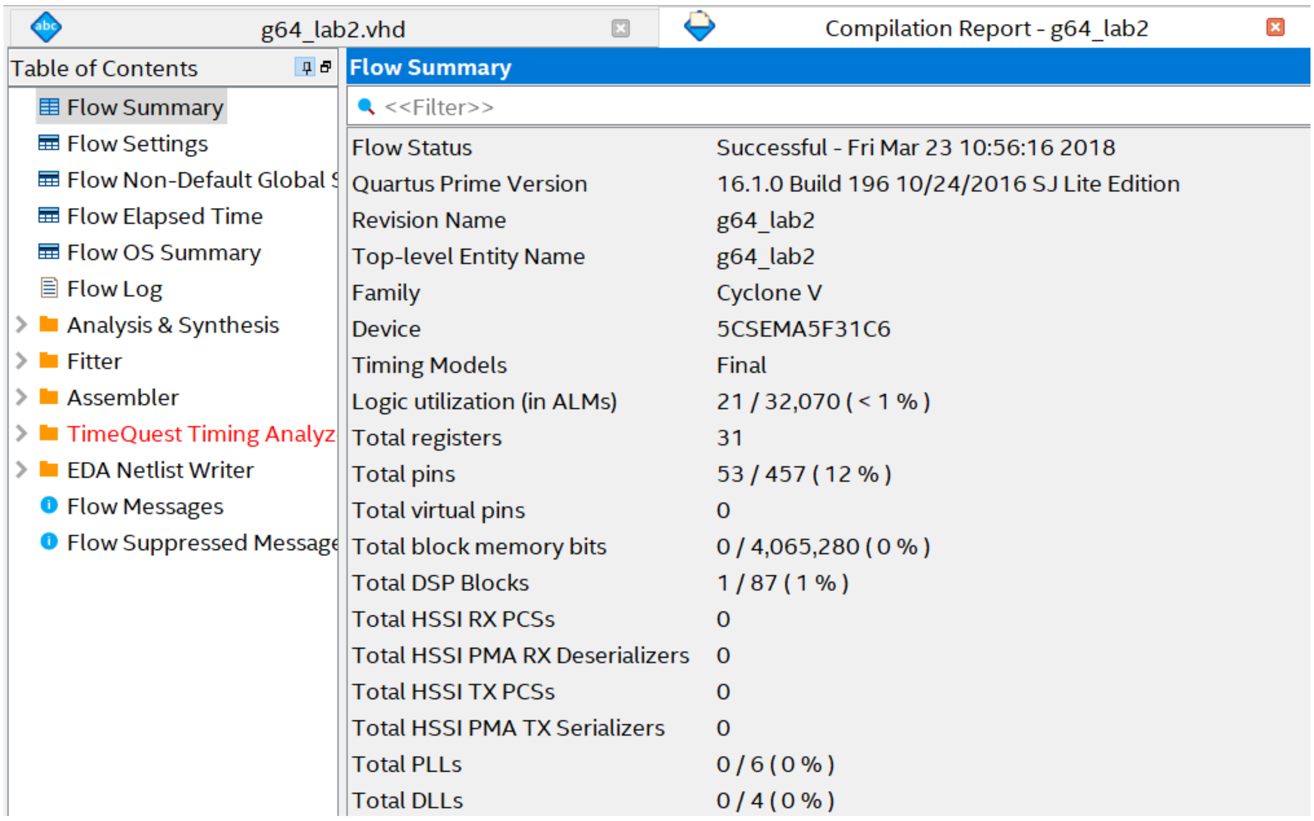


Fig. 1 Testbench Simulation of the MAC unit using ModelSim

Resource Utilization

Compilation Report & Chip Planner



Flow Summary

Flow Status	Successful - Fri Mar 23 10:56:16 2018
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	g64_lab2
Top-level Entity Name	g64_lab2
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	21 / 32,070 (< 1 %)
Total registers	31
Total pins	53 / 457 (12 %)
Total virtual pins	0
Total block memory bits	0 / 4,065,280 (0 %)
Total DSP Blocks	1 / 87 (1 %)
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0 / 6 (0 %)
Total DLLs	0 / 4 (0 %)

Fig. 3 Compilation Report: Flow Summary

Analyzing the flow summary in the compilation report, it can be seen that 31 registers were used in total, which is a pretty low-profile MAC unit implementation as the MAC unit only consists simple multiplier and adder logic. The dark blue on the chip in the Chip Planner (Fig. 4) shows the areas of the chip where resources are used by the MAC unit. This design clearly does not use many resources and thus, a much smaller chip could have been used.

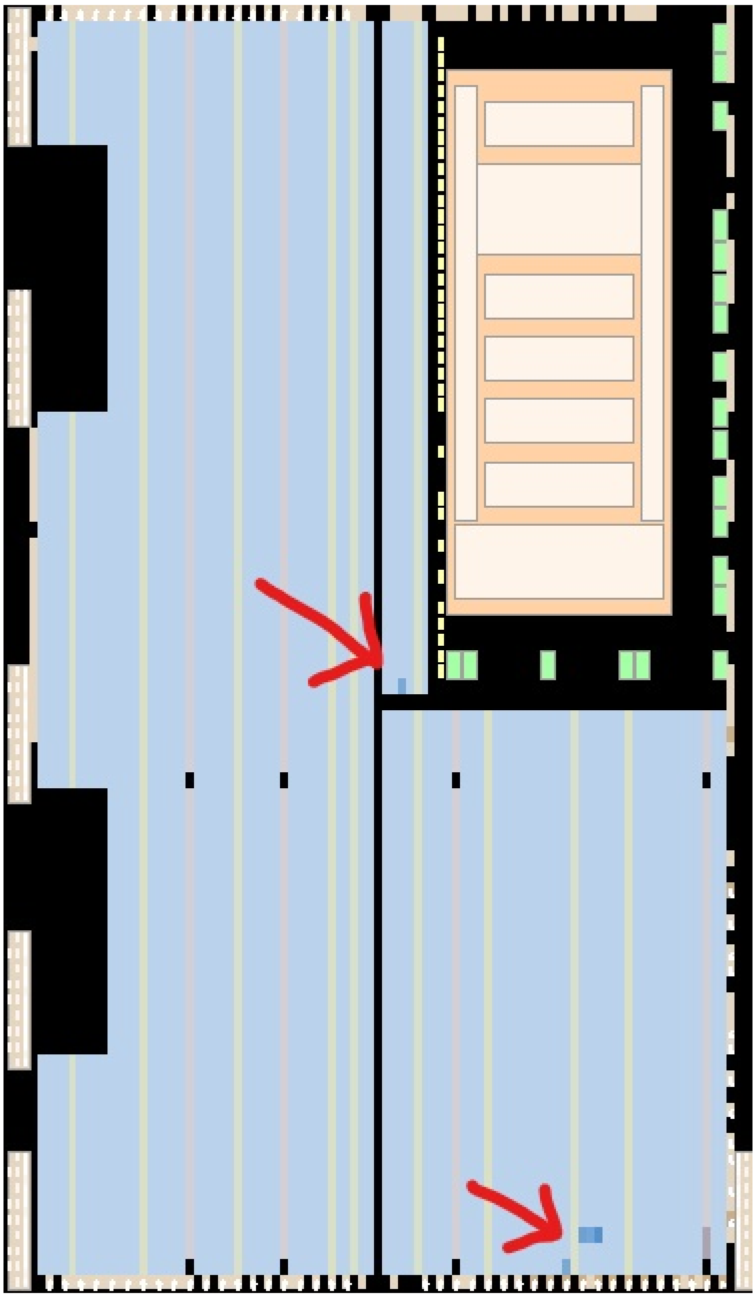


Fig. 4 Chip Planner

RTL View

Fig. 5 below is the Register Transfer Level (RTL) view of the MAC design. Comparing this to the diagram given in the manual (Fig. 6), it can be seen that the core logic has the same resources. One multiplier, to multiply the two inputs, an adder to add the product of the inputs to the accumulated total, and a register to store the result (with the output feeding back to the adder).

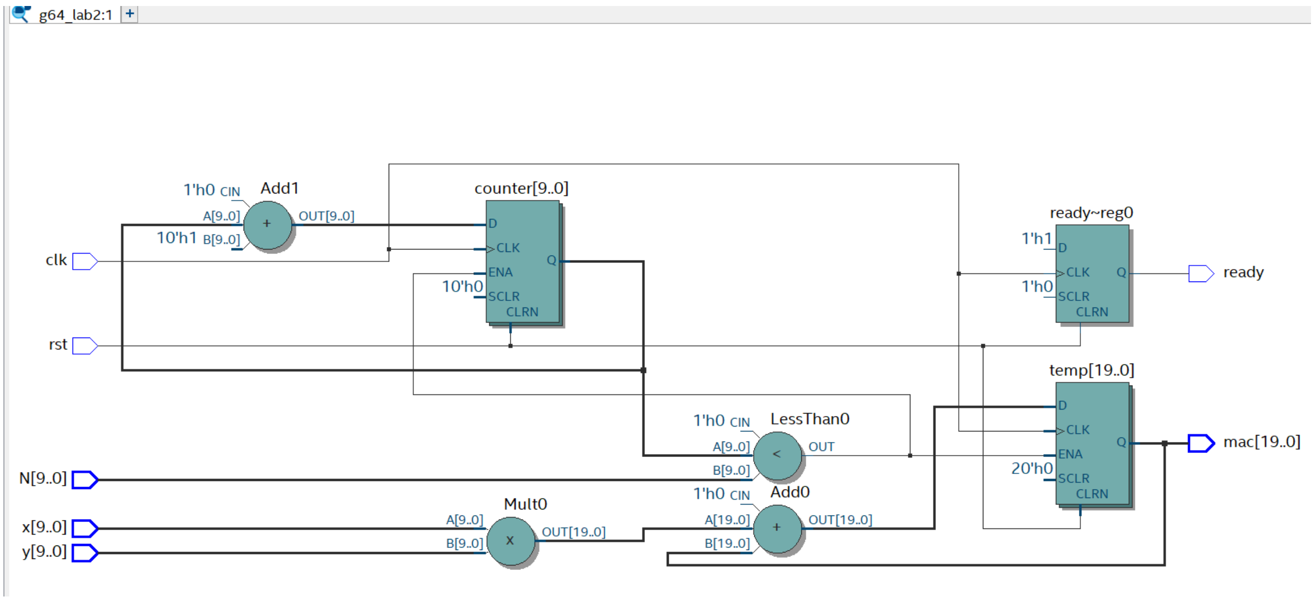


Fig. 5 RTL View of the Circuit

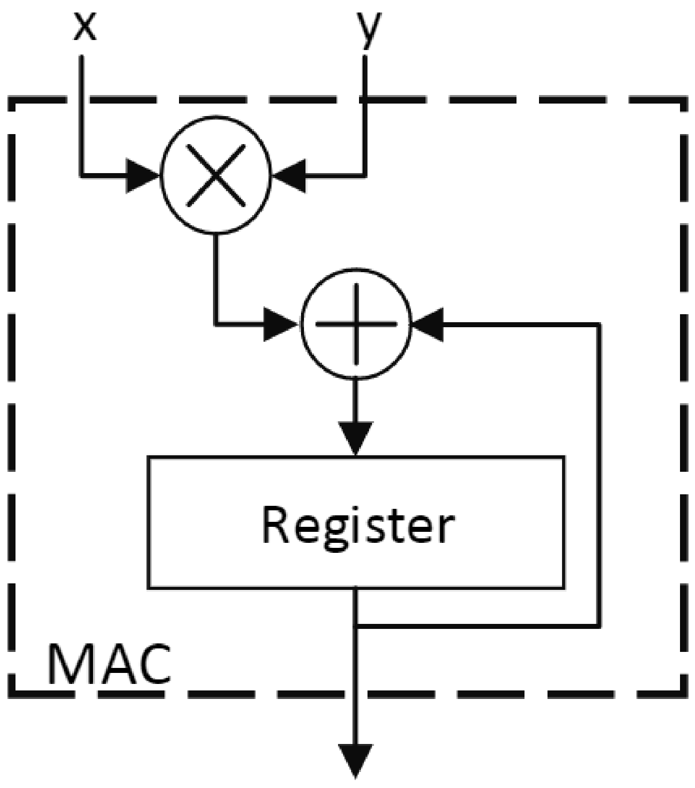
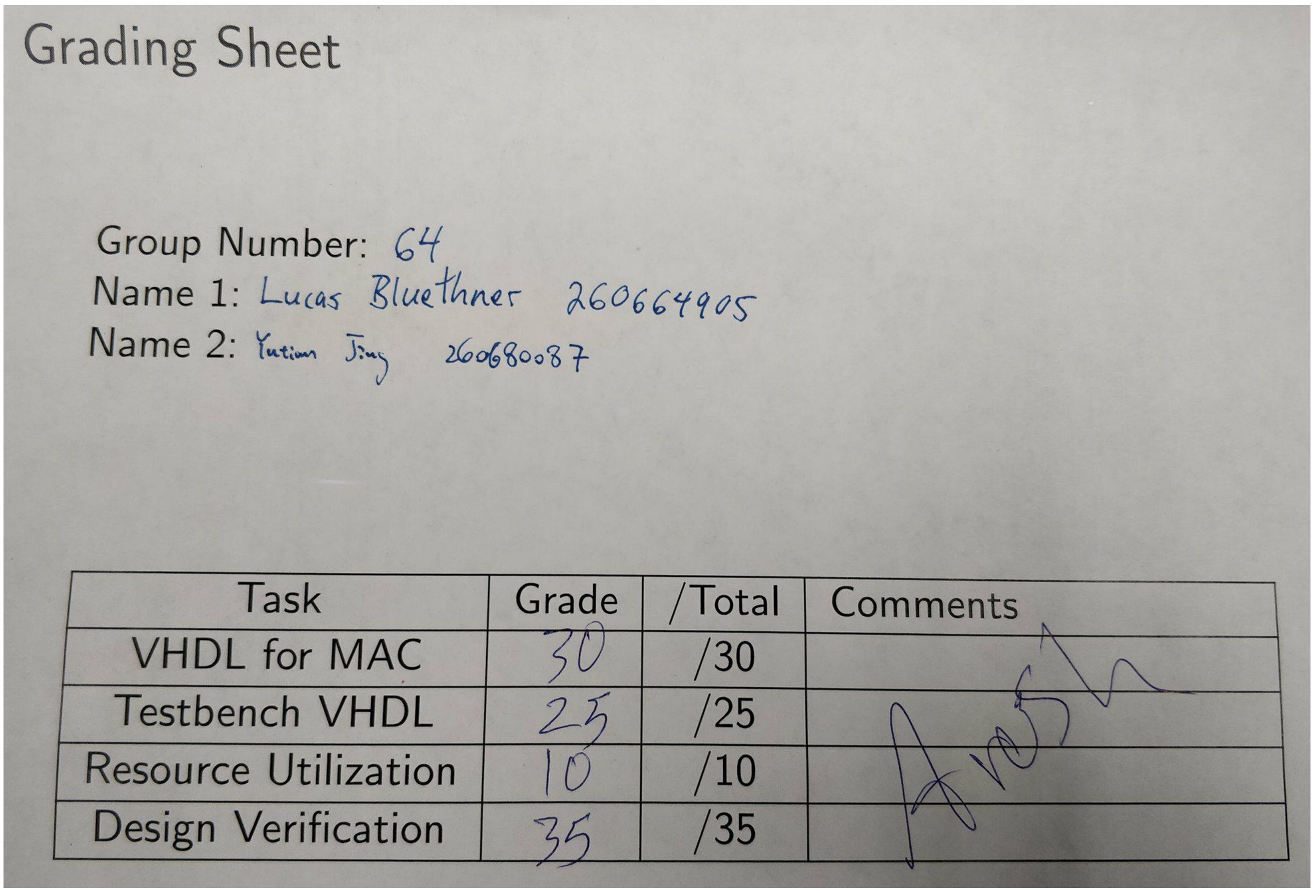


Fig. 6 The high-level architecture of the MAC unit

Conclusion

In this lab we learned the basics of fixed-point representations, VHDL testbench creation, and functional verification using ModelSim. We wrote a program in C to convert floating-point numbers to fixed-point representation, VDHL code to implement a multiplication-accumulation (MAC) unit (using a fixed-point representation), and finally, we wrote code for a testbench which successfully validated our MAC design in ModelSim.

Grading Sheet



Grading Sheet

Group Number: 64
Name 1: Lucas Bluethner 260664905
Name 2: Yutian Jing 260680087

Task	Grade	/Total	Comments
VHDL for MAC	30	/30	[illegible]
Testbench VHDL	25	/25	
Resource Utilization	10	/10	
Design Verification	35	/35	

Fig. 7 Grading Sheet