COL215, SUMMER 2023

HARDWARE ASSIGNMENT 2 REPORT

Jakharia Rishit (2022CS11621) **ANUP LAL NAYAK (2022CS51827)**

INTRODUCTION

The two parts of this assignment were –

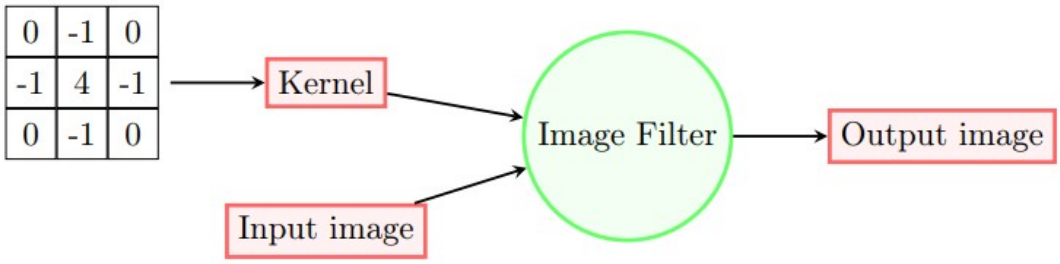
a) Design and test compute unit, implementation of MAC unit (simple or optimized) for image filtering operation.
b) Implement FSM (simple or optimized) and integrate it with the hardware design.

PROBLEM DESCRIPTION AS UNDERSTOOD

- A 64*64 image in taken as input provided in row-major format along with a 9*9 filter also provided as a .coe file in the row-major format
- We have to perform matrix operations to filter the image which involves filtering and normalisation
- Then We need to implement as FSM to control the MAC unit
- The output pixels shall now be displayed on the monitor using the vga port

BLOCK DIAGRAMS

Figure 1: Overview of task: filtering



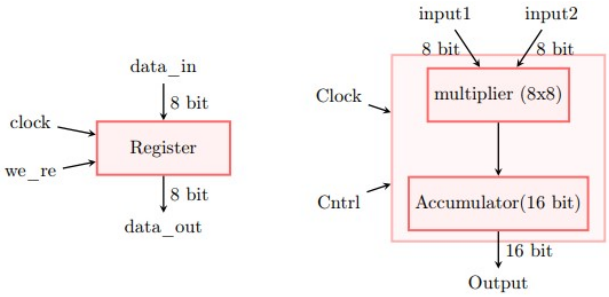


Figure 4: Register and MAC unit

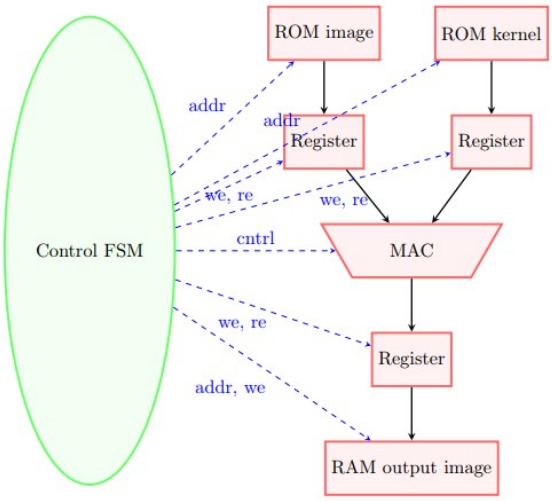


Figure 5: Overview of control path

STRUCTURE ALONG WITH EXPLANATION OF PARTS OF VHDL CODE:

$$O(i,j) = a*I(i-1,j-1) + b*I(i-1,j) + c*I(i-1,j+1)$$
$$+d*I(i,j-1) + e*I(i,j) + f*I(i,j+1)$$
$$+g*I(i+1,j-1) + h*I(i+1,j) + i*I(i+1,j+1)$$

- Case I:

$$O(0,1) = d*I(0,0) + e*I(0,1) + f*I(0,2)$$
$$+g*I(1,0) + h*I(1,1) + i*I(1,2)$$

- Case II:

$$O(1,1) = a*I(0,0) + b*I(0,1) + c*I(0,2)$$
$$+d*I(1,0) + e*I(1,1) + f*I(1,2)$$
$$+g*I(2,0) + h*I(2,1) + i*I(2,2)$$

a) In filtering, we are first using 9 clock cycles to read the filter.coe file kernel and store its values in f1, f2, f3.. f9. These 9 clock cycles are depicted by the changing value of j in our code
b) For each output pixel, we are using 9 clock cycles (depicted by the signal k) to read the image.coe file indexes which are required for the calculation of ith output pixel
c) Below is an excerpt from the code showing that after j exceeds 9, that is after the .coe file is read, we move to the image.coe reading, also, given the value of k, we are handling the corner cases by modifying the value of i, that is the index of radress variable
d) And as shown in the 3rd figure, we calculate the value "calc" after the assing of 9 clock cycles

```
IF (j < 10) THEN
    IF (j = 0) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f0 <= to_integer(signed(filterdata));
        j <= j + 1;
    ELSIF (j = 1) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f1 <= to_integer(signed(filterdata));
        j <= j + 1;
    ELSIF (j = 2) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f4 <= to_integer(signed(filterdata));
        j <= j + 1;
    ELSIF (j = 3) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f7 <= to_integer(signed(filterdata));
        j <= j + 1;
    ELSIF (j = 4) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f2 <= to_integer(signed(filterdata));
        j <= j + 1;
    ELSIF (j = 5) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f5 <= to_integer(signed(filterdata));
        j <= j + 1;
    ELSIF (j = 6) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f8 <= to_integer(signed(filterdata));
        j <= j + 1;
    ELSIF (j = 7) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f3 <= to_integer(signed(filterdata));
        j <= j + 1;
    ELSIF (j = 8) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f6 <= to_integer(signed(filterdata));
        j <= j + 1;
    ELSIF (j = 9) THEN
        filteraddress <= STD_LOGIC_VECTOR(to_unsigned(j, 4));
        f9 <= to_integer(signed(filterdata));
        j <= j + 1;
    END IF;

END IF;
```

```
IF (j > 9) THEN
    IF k = 0 THEN
        IF (new_sig >= 0 AND new_sig < 4096) THEN

            IF type1 = '0' THEN
                rdaddress <= STD_LOGIC_VECTOR(to_unsigned(new_sig, 12));
                type1 <= '1';
            ELSE
                a4 <= to_integer(unsigned(data));
                type1 <= '0';
                new_sig <= i;
                k <= 1;
            END IF;
        ELSE
            a4 <= 0;
            new_sig <= i;
            k <= 1;
        END IF;
```

```
IF k = 10 THEN
    IF type1 = '0' THEN
        calc <= f1 * a1 + f2 * a2 + f3 * a3 + f4 * a4 + f5 * a5 + f6 * a6 + f7 * a7 + f8 * a8 + f9 * a9;
        type1 <= '1';

    ELSE
        type1 <= '0';
        IF (to_integer(signed(max)) < calc) THEN
            max <= STD_LOGIC_VECTOR(to_signed(calc, 20));
        END IF;

        IF (to_integer(signed(min)) > calc) THEN
            min <= STD_LOGIC_VECTOR(to_signed(calc, 20));
        END IF;
```

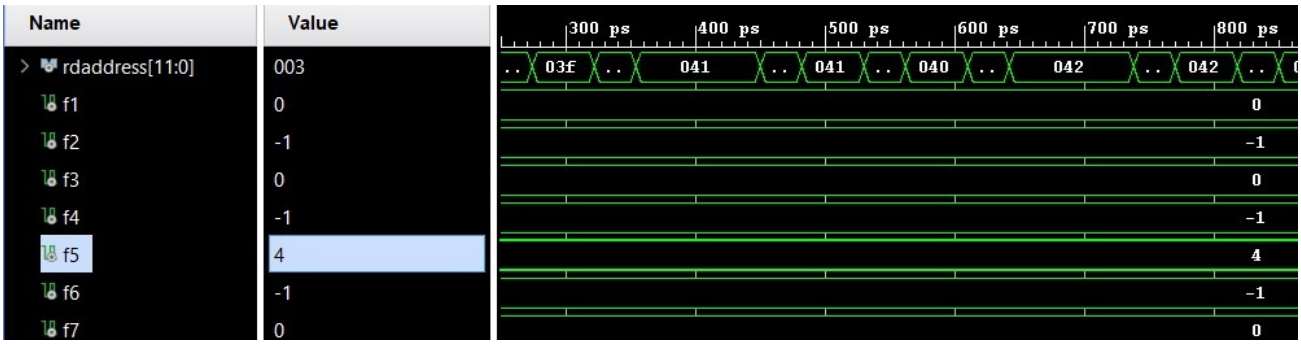
e) After all the output pixel values are calculated, a new normalisation process starts, We use the normalisation formula that they have given and write the normalised value in ram

$$New\_I(i,j) = (I(i,j) - old\_min) * \frac{new\_max - new\_min}{old\_max - old\_min} + new\_min$$

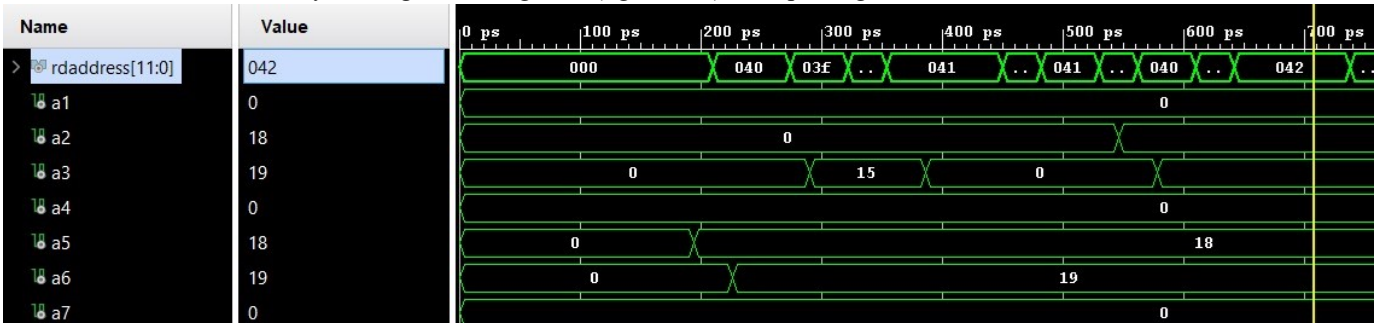
```
IF anup = 1 THEN
    rdaddress <= STD_LOGIC_VECTOR(to_unsigned(i, 12));
    anup <= anup + 1;
ELSIF anup = 2 THEN
    normValue <= (to_integer(signed(data_out)) - to_integer(signed(min))) * 255/difference;
    wr <= '1';
    anup <= anup + 1;
ELSE
    rdaddress <= STD_LOGIC_VECTOR(to_unsigned(i, 12));
    calculated <= STD_LOGIC_VECTOR(to_unsigned(normValue, 20));
    wr <= '0';
    anup <= 1;
END IF;
```

SIMULATION SCREENSHOTS:

a. Correctly reading from kernel and finding values of f1,f2.. f9



b. Correctly reading from image.coe (lighthouse) and updating data1, data2...data9 values



c. Correctly calculating minimum and maximum for normalisation calculation