COL215 - Hardware Assignment 3

Submission Deadlines: [PART 1: 7th November 2023] [PART 2: 14th November 2023]

1 Introduction

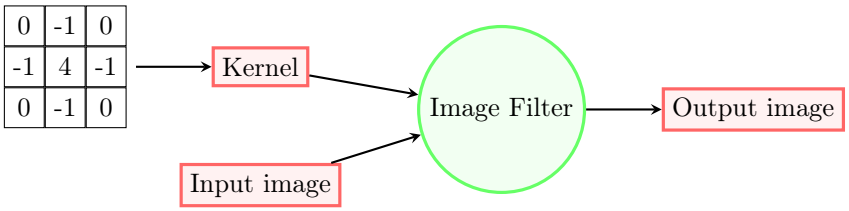
The objective of the assignment is: **implementation of a 3x3 image filtering operation**, which is an extension of the previous hardware assignment. The components involved are memory elements (RAM, ROM and registers), compute unit (filter operation), and VGA controller.

Please note that the diagrams in this document are for illustration purposes and explanation of the problem statement. They do not represent the exact design to be implemented. Design choices can vary across groups and should be carefully explained in the assignment report.

2 Problem Description

Given an image of size **64x64** and an image kernel of size **3x3**, your task is to perform an image filtering operation and display the resultant image. An overview of the problem is shown in Figure 1.

Figure 1: Overview of task: filtering



- Input image size is 64x64 and will be provided in the COE file (8 bit binary unsigned). The image should be stored in 1-D array format in the block RAM, starting from address 0 (0000_{16}) in row major format.
- Kernel size is 3x3 and will be provided via the COE file. It can contain negative values in 2's complement form. (8 bit binary).

You are allowed to use the RAM/ROM and VGA VHDL design code from Hardware Assignment 2.

3 Basics of Image Filtering

An 8-bit grayscale image is represented as a 2-D matrix, each element representing a pixel value between 0-255, as shown in Figure 2. Our objective is to perform an image filtering operation using a 3x3 filter (shown in red in Figure 2). The output image pixel value is the sum of element-wise multiplications between input image pixels and the corresponding kernel.

3.1 Steps to calculate the filtered image

- In performing the filter operation, we stride along the row of the image, and compute the resultant pixels of the output image.
- The output pixel value $O(i,j)$ at location (i,j) is computed as the sum of element-wise multiplications between kernel value and input image pixel, as shown in the equation below. The input image pixel at location (i,j) is denoted $I(i,j)$.

$$\begin{aligned} O(i,j) = a*I(i-1,j-1) + b*I(i-1,j) + c*I(i-1,j+1) \\ +d*I(i,j-1) + e*I(i,j) + f*I(i,j+1) \\ +g*I(i+1,j-1) + h*I(i+1,j) + i*I(i+1,j+1) \end{aligned}$$

  The above computation generates image indices that may be outside the [0..255] range for border pixels. Assume pixel values 0 for such positions. For example:

  - Case I:

$$\begin{aligned} O(0,1) = d*I(0,0) + e*I(0,1) + f*I(0,2) \\ +g*I(1,0) + h*I(1,1) + i*I(1,2) \end{aligned}$$

  - Case II:

$$\begin{aligned} O(1,1) = a*I(0,0) + b*I(0,1) + c*I(0,2) \\ +d*I(1,0) + e*I(1,1) + f*I(1,2) \\ +g*I(2,0) + h*I(2,1) + i*I(2,2) \end{aligned}$$

3.2 Image normalization

You need to perform an *image normalization* step to ensure that the output pixel values fit within 0-255, so that the output image is also stored in the 8-bit unsigned format.

In linear normalization, for each pixel, we perform a scaling according to the equation 1, where old_max and old_min are minimum and maximum pixel values in the output image, new_max and new_min is the new range of pixel value in the final image:

$$New\_I(i,j) = (I(i,j) - old\_min) * \frac{new\_max - new\_min}{old\_max - old\_min} + new\_min \tag{1}$$

After computing the output image, we need to search for the minimum and maximum pixel values, and normalize the pixel values based on these values. For the given normalization, the new minimum is 0 and new maximum is 255, leading to the computation shown in Equation 2.

$$New\_I(i,j) = (I(i,j) - min) * \frac{255 - 0}{max - min} + 0 \tag{2}$$

0	-1	0
-1	4	-1
0	-1	0

	(0,0)	(0,1)	(0,2)	(0,3)								(0,64)
	10	10	10	100								
(1,0)	10	10	10	100								
(2,0)	10	10	10	100								
(3,0)												
(64,0)												

	(0,0)	(0,1)	(0,2)	(0,3)					(0,64)
	20	10	-80						
(1,0)	10	0	-90						
(2,0)									
(3,0)									
(64,0)									

Figure 2: Illustration to perform 1D operation on grayscale image

3.3 Components of the Image Filter

The design consists of three parts, (i) Memory, (ii) Compute unit, and (iii) Finite State Machine (FSM). The FSM controls the reading/writing from the ROM/RAM and addition/multiplication in the Compute unit. The Compute unit consists of components such as arithmetic units and multiplexers. For the given design, some alternative designs are shown in Figure 3 - (i) One FSM to control both memory read/write and compute unit. (ii) Two separate FSMs: FSM1 for memory operations and FSM2 for the compute unit, with a synchronization/interaction between the two FSMs.

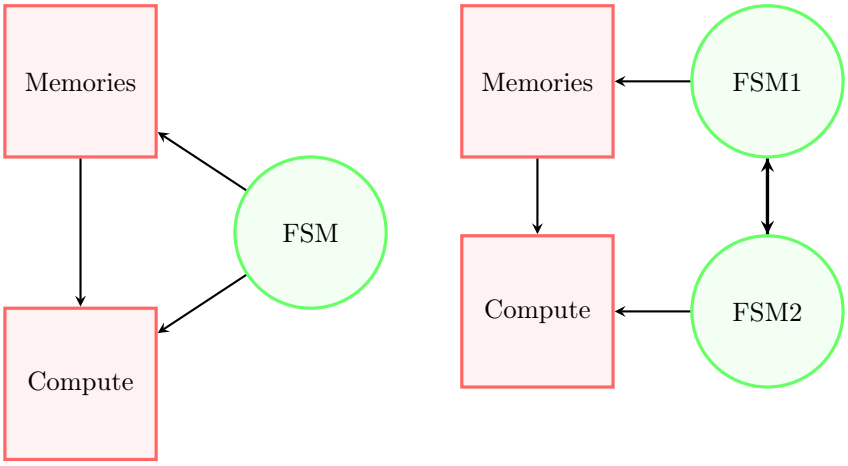


Figure 3: Alternative designs

The overall design would consist of the following:

1. A Multiplier-Accumulator block (MAC): to perform element wise multiplication and accumulation to get one output pixel
2. A Read-Write Memory (RWM, more popularly known as RAM or Random-Access Memory) - Use this memory to store the output.
3. A Read-Only Memory (ROM) - Use this memory to store the input matrix and image kernel.
4. Registers - Use these to store temporary results across clock cycles
5. An overall circuit that stitches all modules together to perform the filtering operation.

All the components are described in the Figure 5, and the block diagram for MAC and register block are expanded in Figure 4.

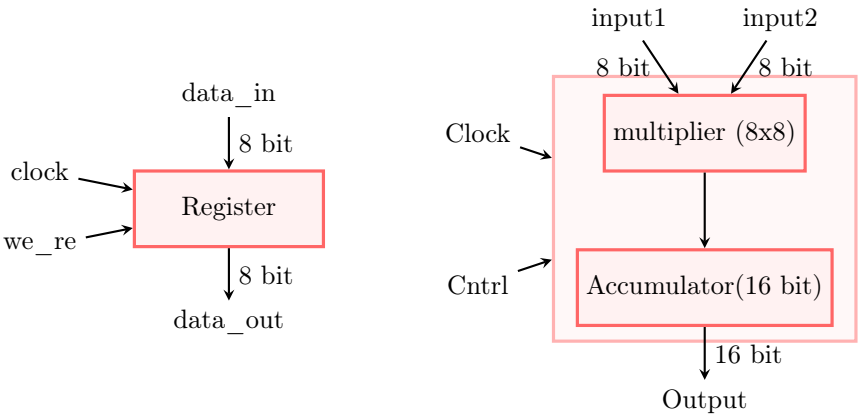


Figure 4: Register and MAC unit

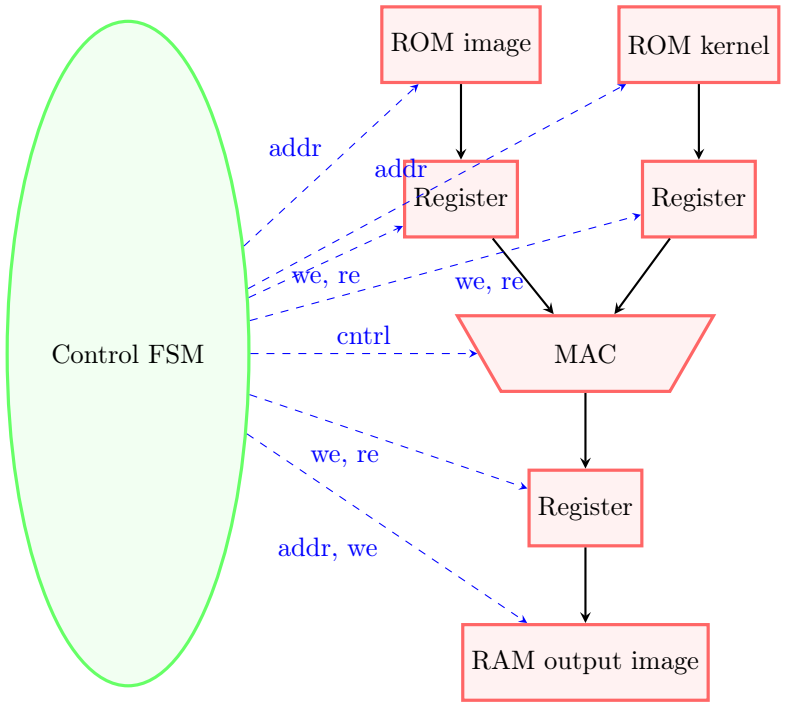


Figure 5: Overview of control path

3.4 Execution time analysis for filtering operation

Let us analyse the required execution time. Based on the simple implementation and using the design in Figure 3, we perform the following estimation.

1. The input pixel is read from ROM, and, in parallel, the first kernel value is read from kernel ROM. This requires 1 clock cycle.

2. Both inputs are fed to the MAC, which performs the multiplication and addition operations, and adds the result to the accumulator register. This requires 1 clock cycle.

3. The above steps are repeated for the next 8 pixels of the kernel. Therefore, to calculate one output pixel, a total of 9 clock cycles are required to read the input values, and 9 clock cycles are required to perform the computation.

Attempt to make the filtering operation faster. As a hint, notice the overlap in Figure 6 between elements of the input image fetched in two successive filtering operations.

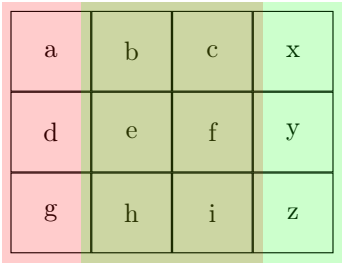


Figure 6: Overlap in filtering operation

3.5 Creating Finite State Machines in VHDL

For the control path, you will need to implement an FSM to control the MAC unit and perform data transfers between ROM/RAM and the local registers. In this section, we illustrate how to implement an FSM in VHDL.

An example FSM shown in figure 7. Its job is to control the operation of a programmable adder/subtractor by sending a signal that instructs the component to ADD or SUBTRACT. The FSM consists of 2 states, ADD and SUB, along with input flags M (to change the operation) and DONE (to know when the current operation is completed); and output flag `cntrl_add_sub` to indicate the operation in the programmable adder/subtractor.

The FSM, specified in VHDL, consists of two parts:

1. Sequential block: In this block, the state is updated on the clock edge or initialized to a default state if the reset flag is set HIGH.

2. Combinational Block: This consists of the logic to update the next state based on the flag values and the current state.

The following is the VHDL template for implementing an FSM. You can see that the state of the system is captured in the Sequential block and updated every positive clock edge, and the work performed in one clock cycle is indicated/controlled in the Combinational block.

```
entity FSM IS
    Port ( M : in STD_LOGIC;
           Done : in STD_LOGIC;
           clk    : in  STD_LOGIC;
           reset : in STD_LOGIC;
           cntrl_add_sub : out STD_LOGIC;
         );
end FSM;
architecture machine of FSM IS
    type state_type is (ADD, SUB);
    signal   cur_state         : state_type := ADD;
    signal   next_state         : state_type := ADD;
begin
   --Sequential block
```

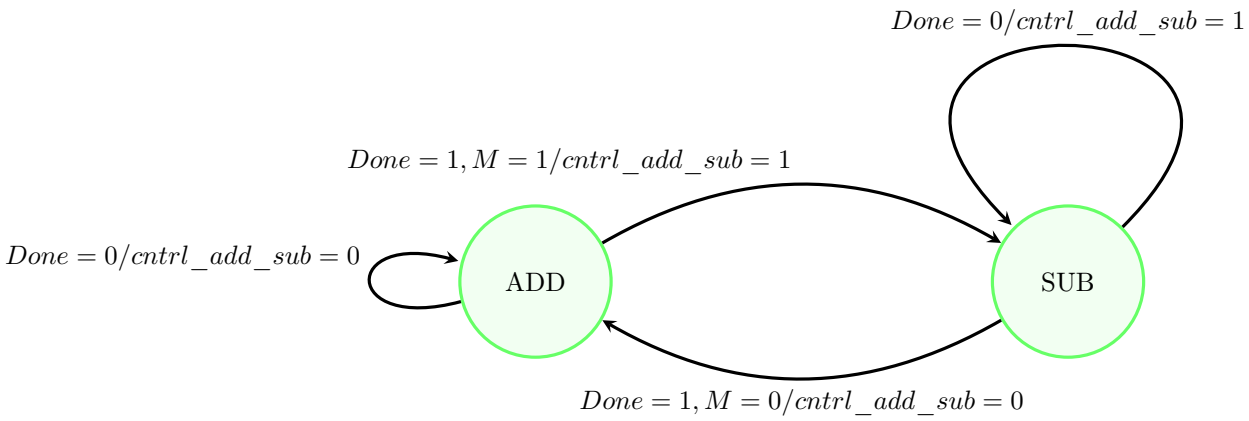


Figure 7: FSM for adder/subtractor

```
   process (clk, reset)
   begin
      if (reset = '1') then
         cur_state <= ADD;
      elsif (clk'EVENT AND clk = '1') then
         cur_state <= next_state;
      end if;
    end process;
    --Combinational block
    process (cur_state, M, Done)
    begin
        next_state <= cur_state;

        case cur_state is
            when ADD =>
                if Done = '1' and M = '1' then
                    next_state <= SUB;
                    cntrl_add_sub <= '1';
                elsif Done = '0' then
                    next_state <= ADD;
                    cntrl_add_sub <= '0';
                end if;
            when SUB =>
                if Done = '1' and M = '0' then
                    next_state <= ADD;
                    cntrl_add_sub <= '0';
                elsif Done = '0' then
                    next_state <= SUB;
                    cntrl_add_sub <= '1';
                end if;
        end case;
    end process;
end machine;
```

4 Submission and Demo Instructions

Since this assignment will span over multiple weeks, your progress in each week will be evaluated as described below.

1. **Make sure that you have a working system first before optimizing it. Another golden rule: Always ensure that a simulation is working correctly before testing on the FPGA board.**
2. **PART 1. Week 1-2 (23rd October - 4th November 2023): Design and test compute unit, implementation of MAC unit (simple or optimized) for image filtering operation.**
3. **PART 2. Week 3 (5th November 2023 - 11th November 2023): Implement FSM (simple or optimized) and integrate it with the hardware design.**
4. You are required to submit a zip file containing the following on Gradescope:
    - VHDL files for all the designed modules.
    - Simulated waveforms with test cases of each subcomponent and complete design.
    - A short report (1-2 pages) explaining your approach. Include block diagram depicting the modules. And short summary about functionality of each module.

5 Resources references

- IEEE document: https://ieeexplore.ieee.org/document/8938196
- Basys 3 board reference manual: https://digilent.com/reference/_media/basys3:basys3_rm.pdf
- Online VHDL simulator: https://www.edaplayground.com/x/A4