COL 215

HARDWARE ASSIGNMENT 3

Hemanshu Garg 2022CS11090 Poojan Shah 2022CS11594

**OBJECTIVES**

To implement a 3X3 image filtering operation on a 64X64 image file and to display the result obtained

**MODULES USED**

The complete design id made up of the following main components :

* Memories (ROM, RAM)
* Compute Unit (MAC,Registers)
* VGA Controller
* FSM

**MEMORY ELEMENTS**

We used vivado’s BRAM (Block RAM) IP for storing both the image and filter -kernel in the ROM, as well as for storing the final result in the RAM which is fed to the VGA Controller. We chose the BRAM architecture in favour of the Distributed Memory Generator due to the following reasons:

* Block RAM provides synchronous read-write operations while Distributed RAM does not, hence it is easy to synchronize the complete design.
* BRAM does not use any extra LUTs(LookUp Tables) and hence is more optimal for resource utilization on the FPGA board

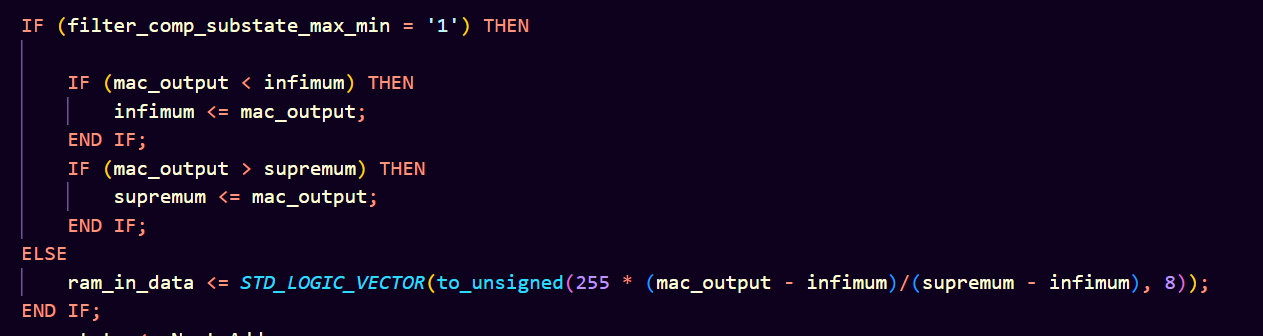
Note that these advantages come at the cost of having a slower design (Distributed RAMs are faster for smaller implementations)

**FILTER OPERATION**

The filter operation consists of two steps : calculating the unnormalized values and then normalizing them. The filtering is actually a convolution between the kernel and the input image :

Since the image is bound to be a 64X64 8-bit grayscale image, we need to normalize our calculations in case they are out of the (0,255) range of values. This is done by using a linear map

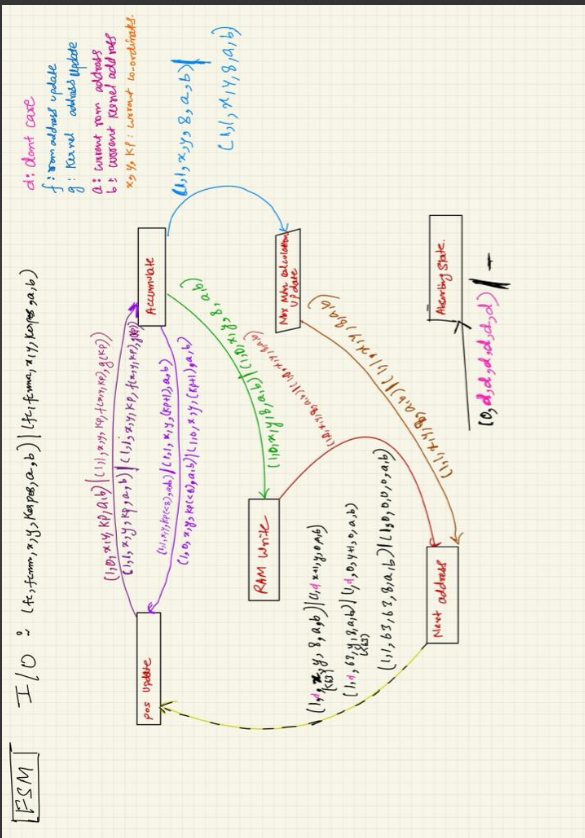
The arithmetic calculations required for the filtering are performed by the MAC(Multiplier Accumulator unit). We also keep track of the minimum and maximum pixel values as we do our calculations, which are later used for normalization.



**FSM**

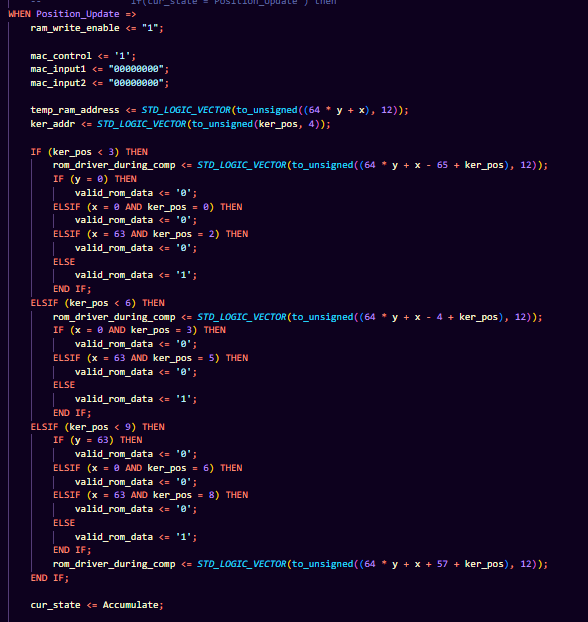
The most crucial part of this assignment is the underlying Finite State Machine which brings together all the components together.

The following diagram shows the logic flow of our design



In the above diagram, represent the position on the image address where we are calculating the filter, represents the position of the current element on the 3X3 kernel matrix. Each transition of the states shown in the above diagram takes exactly one clock cycle. Here is a brief description of the states :

**Position\_Update :** To calculate the filter, we iterate over the neighbours of the point . This is performed in this state by updating the ROM address. The boundary cases are also handled here.



**Accumulate :** The corresponding values from the image ROM and kernel ROM are fed into the MAC unit for performing computations.

A computer screen shot of a program code

Description automatically generated

**RAM Write :** After 9 accumulations to the MAC have been done, the output will be written to the RAM after performing normalization. The state **Accumulate** will go to the **Max Min Update** state depending on the value of the Boolean filter\_comp\_min\_max

A computer code on a black background

Description automatically generated

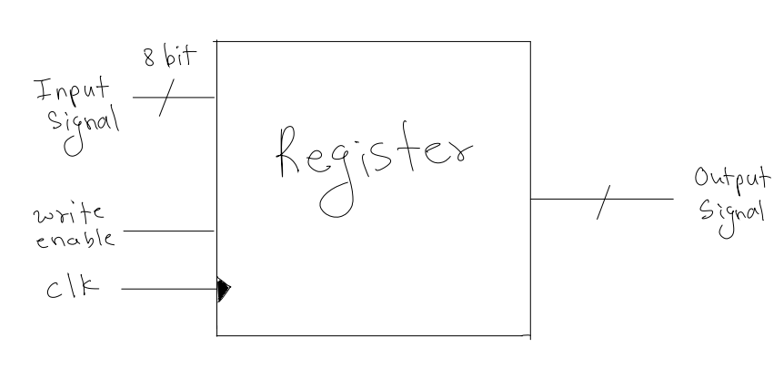
**Next Address :** This iterates over each pixel of the Image ROM.

A computer screen shot of a program code

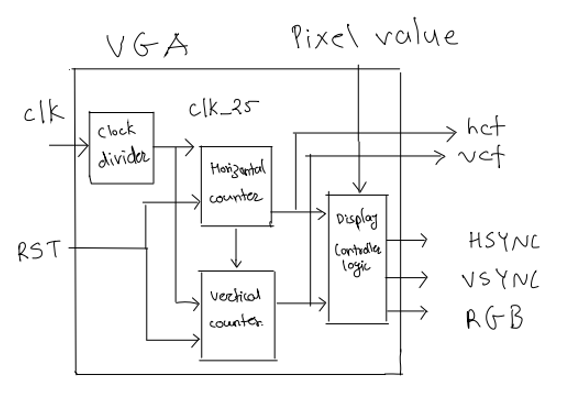
Description automatically generated

**BLOCK DIAGRAMS**

The block diagrams of the individual units which are controlled by the FSM are shown here:

**A white paper with black text

Description automatically generated**

****

**TIMING ANALYSIS**

For the FSM based design, there are exactly 9 transitions, to-and-fro between the Accumulate and Position Update states, and hence we take a total of 18 clock cycles for each computation. We also made a faster version based on hardware assignment 2 which uses the overlap between the filters to make the computation more efficient.

We store the pixel values in nine registers as shown :

A screen shot of a computer program

Description automatically generated

After each computation of the output value, the following transitions are made :

A computer screen with text and images

Description automatically generated

**SMULATIONS AND TEST BENCHES**

The following table provides the simulation waveforms which were used to test and debug our initial design before the actual synthesis procedure :

(The test benches used for their generation are attached in the test\_benches sub directory)

|  |  |
| --- | --- |
| Image ROM |  |
| Kernal ROM |  |
| MAC Unit |  |
| Registers |  |
| Filter Operation |  |
| Normalization (tracking minimum and maximum) |  |

**TESTING**

We tested our final design using various filters and images. Some of the filters that we used included the following : (These coe files are provided on Gradescope as well)

The laplace edge detection filter

)

The invert filter

)

The emboss filter

)

The horizontal edge detection filter

)

The following images were obtained and verified for the laplace edge detection filter:

|  |  |
| --- | --- |
| Original Image | Image After Filter Operation |
|  |  |
|  |  |
|  |  |
|  |  |

The following images show the effect of different kinds of filters on the same image :

|  |  |
| --- | --- |
| Original Image |  |
| L Filter |  |
| I Filter |  |
| E Filter |  |