Image Filtering Hardware Design Assignment Report

Rajarshee Das 2022cs11124

Sahil Kumar 2022cs11100

Introduction:

The objective of this assignment is to implement a 3x3 image filtering operation as an extension of the previous hardware assignment. The assignment involves the use of memory elements (RAM, ROM, and registers), a compute unit (for the filter operation), and a VGA controller. This report documents our approach and the design choices made during the implementation.

Problem Description:

The task involves filtering a 64x64 image using a 3x3 kernel and showcasing the resulting image. The image, stored in a COE file in a 1-D array format within block RAM, and the kernel, also provided in a COE file, must be used for the filter operation. The goal is to execute the filtering process using the specified 3x3 kernel and subsequently adjust the output image to comply with the 8-bit unsigned format (0-255).

Basics of Image Filtering:

The process of image filtering encompasses the multiplication of each pixel in the input image by the corresponding kernel value, followed by the summation of these products to derive the output pixel value. Employing a 3x3 kernel, the operation is applied across the entire image. To address border pixels, a placeholder value of 0 is assumed for positions where the kernel exceeds the image boundary.

Components of the Image Filter:

Our design comprises multiple components:

1. Multiplier-Accumulator block (MAC): Employed for performing element-wise multiplication and accumulation and controlled by appropriate control signals.
2. Read-Write Memory (RAM): Utilized for storing the resulting image.
3. Read-Only Memory (ROM): Stores both the input image and the kernel.
4. Registers: Responsible for storing temporary outcomes during different clock cycles.
5. FSM : Responsible for generating appropriate control signals to activate various components of our program based on the current state and input from the ongoing process.
6. VGA controller : This component is responsible for generating appropriate Hsync,Vsync and RGB values to the monitor.

For Part1:

We have generated a RAM,ROM and MAC in our first week. We have also written the main file which is ultimately controlled by the FSM in the final file though this code was not totally complete and we have to complete it in the second week.

The MAC itself has two components one multiplier and other accumulator.

Multiplier : It is a component which just takes two inputs and gives the output as a multiplication of the two input data.

Accumulator: The accumulator takes data and a control signal and gives the accumulated value as a output signal. The control signal can be used to hold the value or add the input value to the current stored value.

The Final component has a filter unit which was meant to be controlled by the signal form a FSM it reads the input COE file and the kernel file computes the unnormalized values first and after iteration stabilises the maximum and minimum value in the unnormalized data values.

Then the normalisation unit was supposed to be functional after input from the FSM and it recomputes the unnormalized value and normalised it by using already computed maximum and minimum values.

More details about the part 1 can be found in the link provided for Part 1 submission.

Final Implementation:

We've developed an image filter and MAC in VHDL to manage the calculated data values. The image filter retrieves values from the kernel ROM and the image ROM, performing multiplication between the corresponding kernel and pixel values. Our MAC accumulates the calculated pixel values for nine pixels and delivers the output after completing the accumulation of these pixels. Throughout the iteration of the ROM, we maintain maximum and minimum counters, comparing each data value and updating the maximum and minimum values accordingly. We then used FSM logic with 3 states, namely state 0, state 1 and state 2. The entire process stays in state 0 until and unless the old_max and old_min are calculated for the entire filter operation. Then our FSM moves to state 2, where it uses the MAC and registers to do the final filtering along with the clamping of the pixel values and store the pixel values in their corresponding addresses in RAM. Then our process moves to state 2 where we display from RAM using VGA which is a separate component.

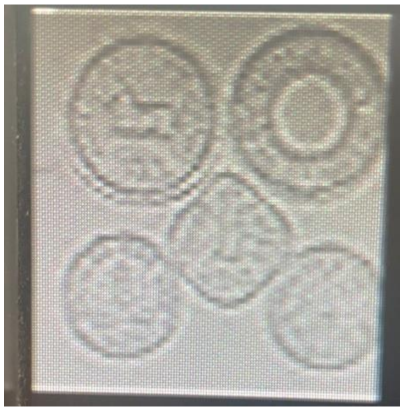
We have chosen to recompute the unnormalized value again in the normalisation process as storing the unnormalized value beforehand would lead to more memory utilization i.e. more hardware utilisation as there is generally a tradeoff between memory and time we tried to optimise memory in this case as the time required for computation is very small the total number of operations being performed at frequencies in megahertz and the number of operations being small as the input COE file in this assignment was given to be of 64*64 if the size would have been more than the time factor would have rosen up very fast and in that case optimising time would have been the priority but as the input is small optimising resource was priority.

The testbenches for the individual component are submitted on the gradescope in the link provided for submission.

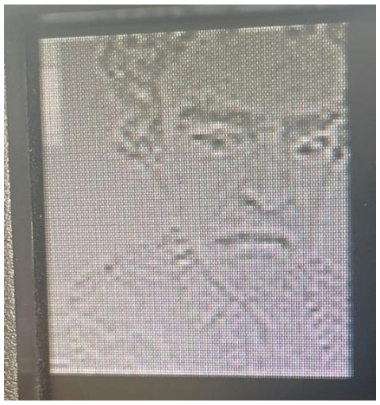
Execution Time Analysis:

We analyzed the required execution time and found that one output pixel requires a total of 40 clock cycles to read input values and perform the computation.This included the time for the output to be stabilised while reading from the rom and the time required for the stabilisation of outputs from other components as the components used by us were sensitive to the rising edge of the clock. We also considered ways to optimize the filtering operation by exploiting overlap between elements of the input image fetched in two successive filtering operations. Basically we used somewhat like sliding window approach.

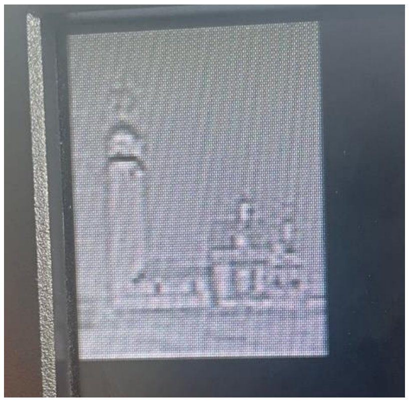
Final Images:



Coins

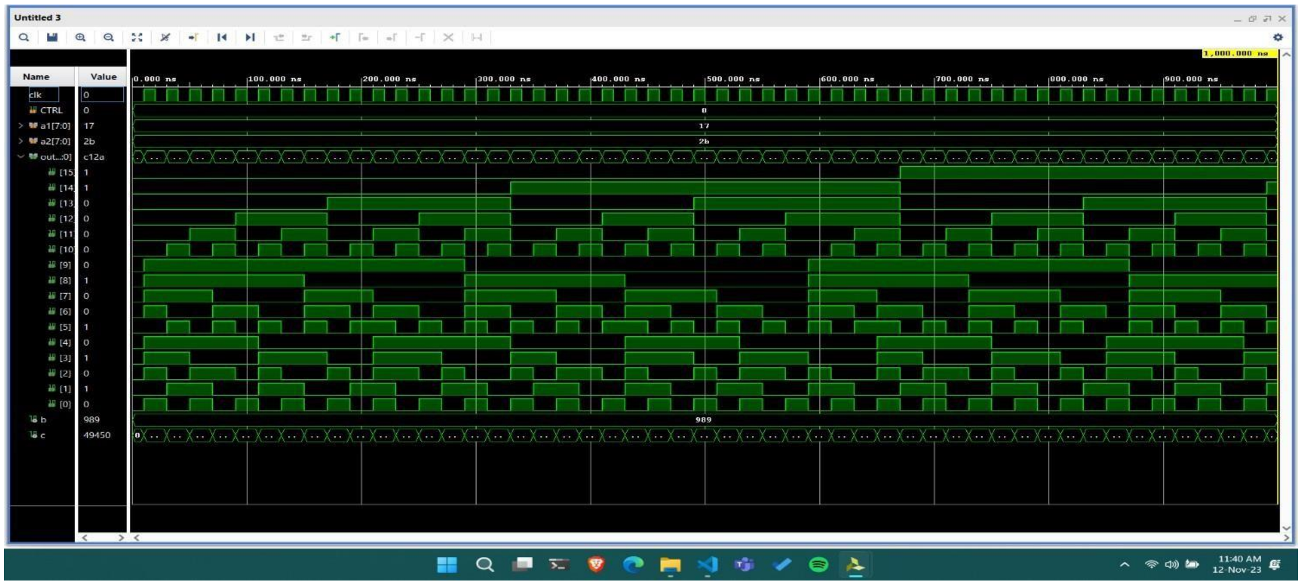


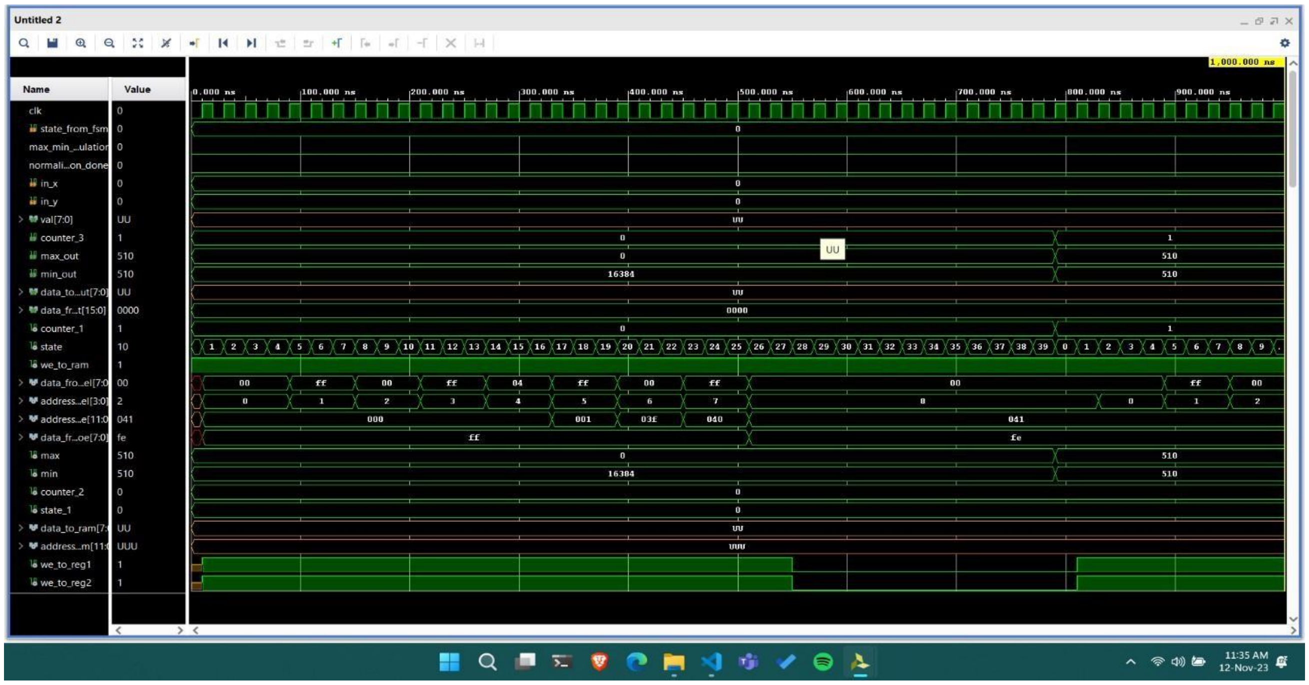
Face

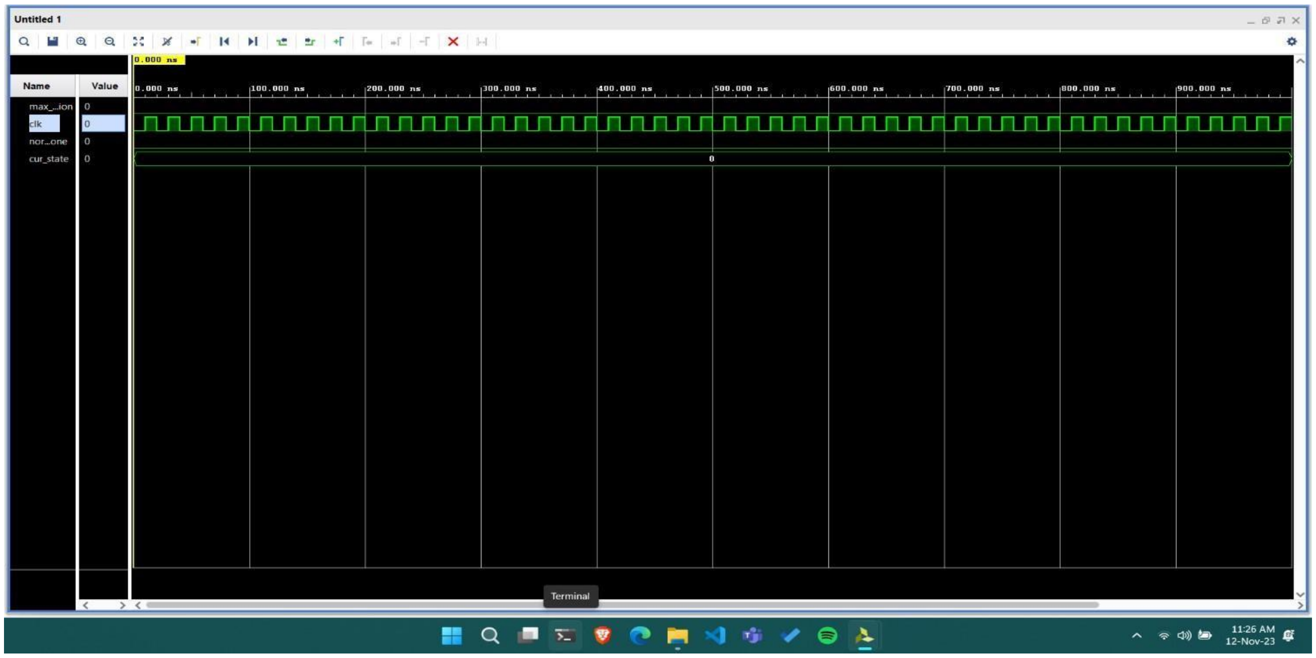


Lighthouse

Simulation Images:



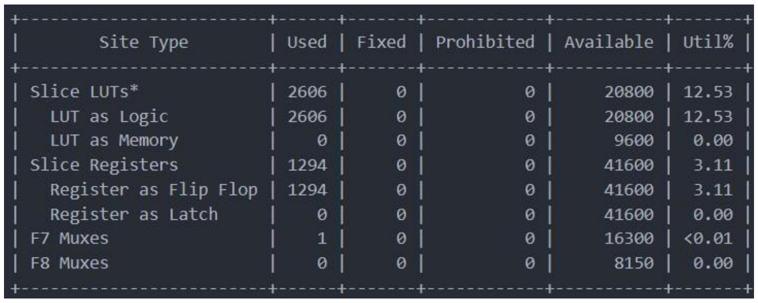




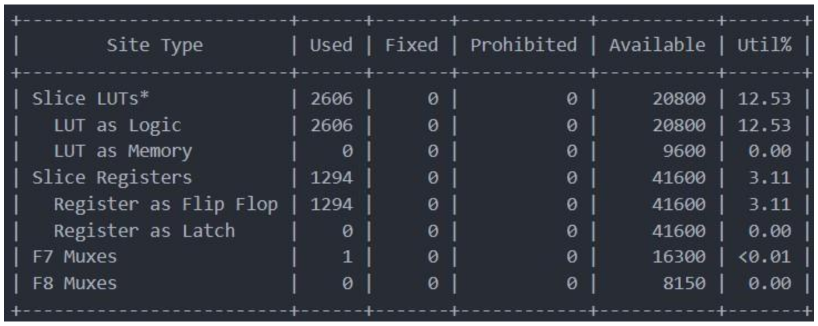
The above simulation images are of each component, so they might seem incomplete but when we combine all the components together then they work perfectly.

Utilization Reports:

Below attached are utilization reports:



```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             | 2606 |     0 |          0 |     20800 | 12.53 |
|   LUT as Logic          | 2606 |     0 |          0 |     20800 | 12.53 |
|   LUT as Memory         |    0 |     0 |          0 |      9600 |  0.00 |
| Slice Registers         | 1294 |     0 |          0 |     41600 |  3.11 |
|   Register as Flip Flop | 1294 |     0 |          0 |     41600 |  3.11 |
|   Register as Latch     |    0 |     0 |          0 |     41600 |  0.00 |
| F7 Muxes                |    1 |     0 |          0 |     16300 | <0.01 |
| F8 Muxes                |    0 |     0 |          0 |      8150 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```



```
+-------------------------+------+-------+------------+-----------+-------+
|        Site Type        | Used | Fixed | Prohibited | Available | Util% |
+-------------------------+------+-------+------------+-----------+-------+
| Slice LUTs*             | 2606 |     0 |          0 |     20800 | 12.53 |
|   LUT as Logic          | 2606 |     0 |          0 |     20800 | 12.53 |
|   LUT as Memory         |    0 |     0 |          0 |      9600 |  0.00 |
| Slice Registers         | 1294 |     0 |          0 |     41600 |  3.11 |
|   Register as Flip Flop | 1294 |     0 |          0 |     41600 |  3.11 |
|   Register as Latch     |    0 |     0 |          0 |     41600 |  0.00 |
| F7 Muxes                |    1 |     0 |          0 |     16300 | <0.01 |
| F8 Muxes                |    0 |     0 |          0 |      8150 |  0.00 |
+-------------------------+------+-------+------------+-----------+-------+
```

The entire utilization report is uploaded along with all the files in Gradescope Submission.

Conclusion:

We successfully implemented a 3x3 image filtering operation using hardware design. We utilized a Multiplier-Accumulator block, RAM, ROM, registers, and FSM logic to create an image filter.