Project Report Digital Systems
Image Processing Toolkit

Abhinav — Hrriday — Sujal — Aditya

April 23, 2024

Contents

1 Introduction **2**

2 Tasks **2**
2.1 Expectations . . . 2
2.1.1 Assigned Tasks . . . 3
2.1.2 Communication . . . 3

3 Strategy **3**

4 Modules Explanation **3**
4.1 Imrx . . . 3
4.2 Padder . . . 3
4.3 Conv . . . 4
4.4 Padder2 . . . 4
4.5 Conv2 . . . 4
4.6 Imadder . . . 4
4.7 Imtx . . . 5
4.8 Topmodule . . . 5

5 Final Model Pipeline **5**

6 Results **6**

7 Errors and Shortcomings **8**

8 Limitations **8**

9 References **8**

Abstract

This project is a part of the course ES 204 here at IITGN aimed at learning Digital Systems. This course is taught by Professor Joycee Mekie. This was our introduction to working with FPGAs and learning circuit design. We had first-hand experience with low-level design working with processor level design, making pipelines and working with micro-operations. Through this course, we delved into the intricate world of digital logic, exploring various aspects such as combinational and sequential logic circuits, minimization techniques, synchronous and asynchronous sequential circuits, and finite state machines. This project primarily focused on implementing image processing on FPGAs, providing us with a practical understanding of how job scheduling actually occurs in circuits like these.

1 Introduction

FPGAs (Field-Programmable Gate Arrays) are ICs designed to be configured or programmed after manufacturing. They are highly flexible unlike fixed architecture ICs which can not be programmed. FPGAs can be highly parellelized which makes them ideal for tasks such as image and video editing. It is greatly used in signal processing for that reason alone. It can provide significant speed ups compared to the traditional sequential processors.

Thus, we use this quality of FPGAs to design a fast image processing toolbox. We designed and wrote Verilog codes for establishing UART communication and perform Image Smoothing and Sharpening on the FPGA itself.

We take an classical image processing approach, using hand-crafted kernels for a convolution based processing approach. Convolution based approaches use kernels to extract higher level features from a raw image.

2 Tasks

2.1 Expectations

Our expectations from this project is an end to end pipeline where we can we can send any image to the FPGA board via a custom python wrapper which is set up with a pre-configured baud rate that is agreed upon both by the board and the transmitter. The image is processed on-device and the final result is communicated back to our transmitting device.

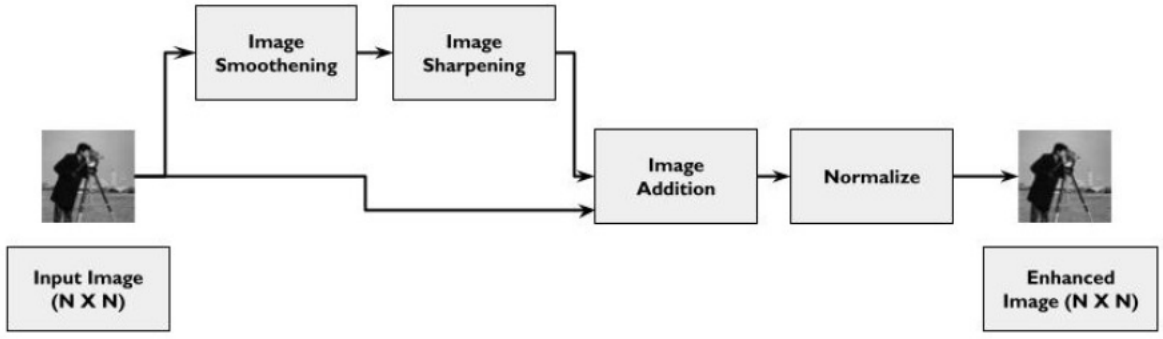


Figure 1: Caption

2.1.1 Assigned Tasks

We have been assigned the following tasks:

1. Design and test all individual blocks.
2. The image size is 128×128. Each pixel of the image is 8 bits.
3. Input/output cannot be an array. So, the Input will be a vector of 131072 ($128 \times 128 \times 8$) bits, which is then to be converted into a two-dimensional matrix inside the module.
4. For Image Smoothing, use a 3×3 Average Filter.
5. For Image Sharpening, use a 3×3 Laplacian Kernel. Ensure proper handling of negative numbers.
6. Image Smoothing and Image Sharpening are convolution operations on the image.
7. Perform addition of sharpened and enhanced image.
8. Finally, normalize the image pixels between 0 and 255 to obtain the final enhanced image.

2.1.2 Communication

Although many methods of communication protocols can be used with a modern FPGA board, we adopted a UART protocol for pedagogical objectives as suggested by Prof. Joycee and our project guide Ruchit Chudasama. UART(Universal Asynchronous Receiver-Transmitter) is a low-speed transmission protocol used for serial data transmission. It is asynchronous, which means that there is no shared clock between the transmitter and the reciever. Instead, the devices synchronize themselves using start and stop bits that are transmitted alongside the data.

3 Strategy

As suggested by Professor Joycee and our guide Ruchit Chudasama

- We are going to approach the pipeline as a work-along of different individual modules and hence work on developing the individual modules first, then create a topmodule to sort of bring all the modules together and control their working.
- We are going to take a "weekly goals" strategy to control the workload and get a hint of progress. And achievability of the tasks.

4 Modules Explanation

4.1 Imrx

The imrx module, short for image receive, is the module we made to receive the image from the computer terminal to the FPGA board using serial UART communication. The module works at a baud rate of 9600 and takes input from the computer terminal serially. The buffer that is used in the module is 8 bits, which when filled, gets written to the blk_mem_gen_0 'top', and the module readies itself to receive the next 8 bits. In such way, the module receives 16384(128*128) words of 8 bits and stores them into the blk_mem_gen_0 'top' block ram.

4.2 Padder

Now that the original image data is received, we need to perform convolution on it. However, there is a size reduction in convolution depending on the size of the kernel. Since we are using a kernel of size 3x3, we would lose 2 rows and 2 columns of the image. Hence we made the padder module to pad the image into a 130X130 image i.e. it takes the 128*128 data front the blk_mem_gen_0 'top' that contains the original image, and pads it to 130x130 and stores it in the blk_meme_gen_1 'padded'. The module extends the top and bottom row by one and the right and left column by one, then puts the corner values the same as the corner values of the original image.

4.3 Conv

The module performs "naive" convolution operation on 130 x 130 padded image (stored in block_mem_gen_1 'padded') using a 3 x 3 kernel to obtain a 128 x 128 convolved image (stored in block_mem_gen_0 'processed'). The module is implemented as a parameterized code that allows for convolution of any (N x N) image with a (p x p) kernel. As is implemented for the pipelined task, the module performs smoothening/blurring of the image using an average 3 x 3 kernel (all ones kernel). The module is implemented as an FSM (Finite State Machine) in which computations are carried out sequentially at a clock frequency slower than the standard 100 MHz of Basys3 FPGA and pixel values (stored linearly in block ram) are accessed using a special index mapping governed by write address.

- The State 0 of the FSM is just an idle state with reset. Unless reset is set low, the FSM stays in State 0.
- The State 1 of the FSM in its entirety computes the sum of the element-wise product of the kernel (all ones kernel) with a subset of padded image (accessed from block_mem_gen_1 'padded').
- The State 2 of the FSM simply divides the sum by 9 (average kernel) and stores the quotient at the appropriate address of the block_mem_gen_0 'processed'. Unless the entire padded image is convolved, the FSM transitions back to State 1 to compute the next convolved pixel value.
- The State 3 of the FSM is a sink state, signaling the completion of convolution.

4.4 Padder2

This module works similar to the padder module discussed above, the only difference is that instead of padding the data of the original image from blk_mem_gen_0 'top', the module pads the data of the processed (smoothened as per our pipeline) image from blk_mem_gen_0 'processed'. The padded data is still stored in the blk_mem_gen_1 'padded'.

4.5 Conv2

The module is similar to conv module described above, performing "naive" convolution operation on 130 x 130 padded image (stored in block_mem_gen_1 'padded') using a 3 x 3 kernel to obtain a 128 x 128 convolved image (stored in block_mem_gen_0 'processed'). As is implemented for the pipelined task, the module performs sharpening of the smooth padded image using a laplacian 3 x 3 kernel. The module is implemented as an FSM (Finite State Machine) with similar structure and index mapping governed by write address.

- The State 0 of the FSM is just an idle state with reset. Unless reset is set low, the FSM stays in State 0.
- The State 1 of the FSM in its entirety computes the signed sum of the element-wise signed product of the laplacian kernel with a subset of padded image (accessed from block_mem_gen_1 'padded').
- The State 2 of the FSM simply stores the sum at the appropriate address of the block_mem_gen_0 'processed'. However, since the sum can be negative or greater than 255 (max pixel intensity), the sum values are clamped to 0 and 255 respectively. Unless the entire padded image is convolved, the FSM transitions back to State 1 to compute the next convolved pixel value.
- The State 3 of the FSM is a sink state, signaling the completion of convolution.

4.6 Imadder

The module effectively performs element-wise addition of pixel values of the original image stored in block_mem_gen_0 'top' and that of image obtained after sharpening the smoothened image stored in block_mem_gen_0 'processed' and stores back to block_mem_gen_0 'processed' after appropriate normalization (division of sum by 2).

4.7 Imtx

The imtx module, short for image transmit, is the module we made to transmit the final processed data back to the computer terminal using serial UART communication. The module works at a baud rate of 9600 and transmits the processed image data to the computer terminal serially. The module transfers one byte (8 bits) serially first, then again refills the data buffer. In such way, the module transmits 16384(128*128) words of 8 bits from the blk_mem_gen_0 'processed' block ram.

4.8 Topmodule

The topmodule is sort of a controller that allows the user to interact with and control the different modules' working. All the three block rams are initialized in the topmodule and thus, it also has the controlling logic for the block rams. For creating the topmodule, after having made the different independent modules, we followed a simple yet effective idea of assuming that the block rams of the independent modules were pulled out of them, so effectively,

- The input to the blockrams would become the outputs of the independent modules.
- The outputs of the block rams would become the inputs of the independent modules.
- The topmodule has the logic control for when the block ram would be accessed by which module (sort of like a mux).

5 Final Model Pipeline

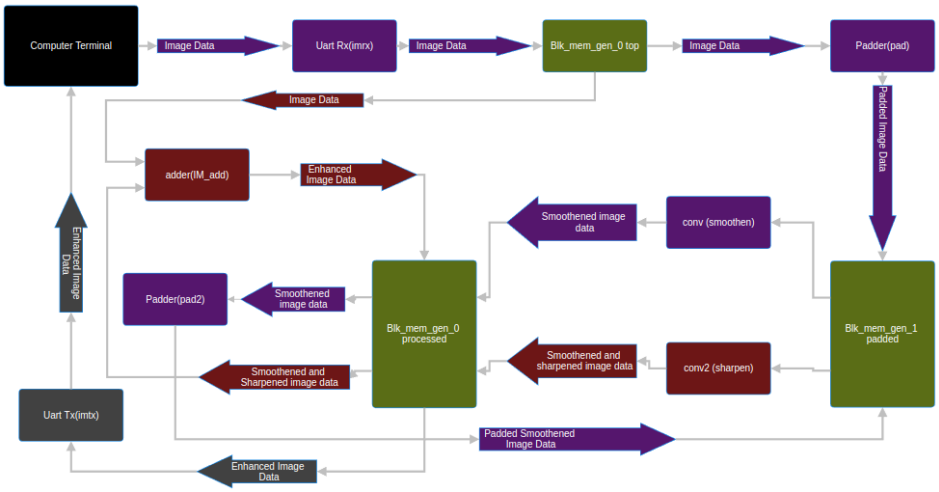


Figure 2: Workflow

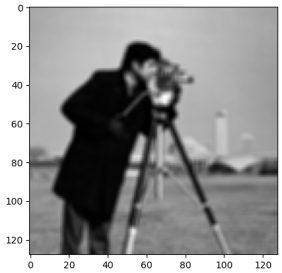
Following the strategy as discussed in the above section. We put together the final pipeline for image enhancement as follows:-

- The image vector is transmitted to the FPGA board through UART communication (done using the imrx module).
- The obtained image is stored in the block ram(called bram_mem_gen_0 'top') of the FPGA (this logic is controlled by the topmodule).
- The image stored in the top bram is the original image.
- This 128X128 image is then padded to 130X130 using the padder module.
- The padded image is stored in another block ram(called bram_mem_gen_1 'padded') of the FPGA (this logic is controlled by topmodule and padder module together).

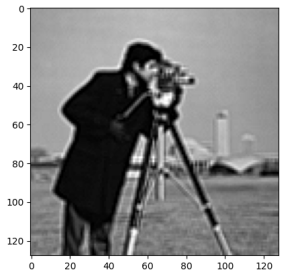
- Smoothing convolution is done on the padded image using the conv module.
- The convolved image (of size 128X128, as the size is reduced when convolving, which is why the image was padded to 130X130) is then stored into the third and the last block ram that we need to use(called blk_mem_gen_0 'processed').
- The image data(128X128) stored in the 'processed' block ram is again padded to 130X130 and stored back in the 'padded' block ram. This is done using the pad2 module
- Sharpening convolution is done on the padded image using the conv2 module.
- The convolved image (of size 128X128) is then again stored back into the 'processed' block ram.
- The image stored in the 'processed' block ram at this point is the original image that was first smoothened, then sharpened.
- The image stored in the 'processed' block ram(smoothened and sharpened) and the 'top' block ram (original) are added together and normalised using the IM_add module.
- The added then normalised image is being stored into the 'processed' block ram.
- The image hence stored in the 'processed' block ram is the enhanced image.
- The image data from the 'processed' block ram is transmitted to the computer from the FPGA board though UART communication (done using the imtx module).

6 Results

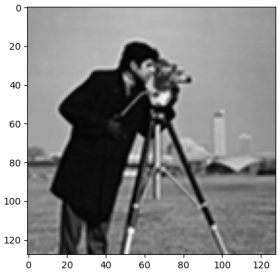
Figure 3: Original Image



(a) Smoothened Image (Avg Kernel)



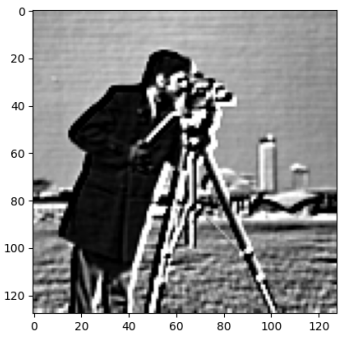
(b) Smoothened and Sharpened Image (Laplacian Kernel)



(c) Enhanced Image (Image Add & Normalize)

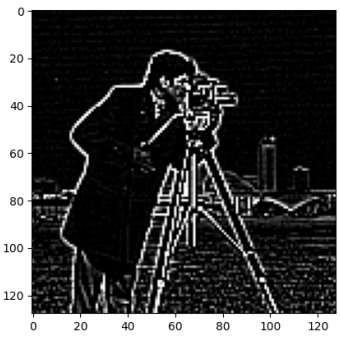
Figure 4: Images through the pipeline

All original project objectives as mentioned in 2.1.1 accomplished!

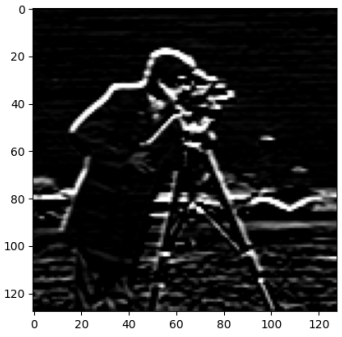


(a) Emboss Filter

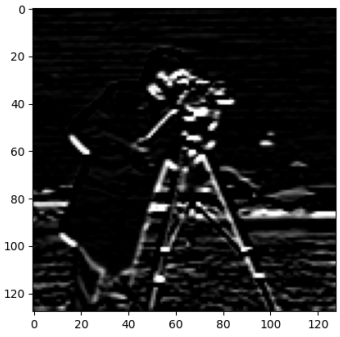
(b) Sharpen (Laplacian) Kernel



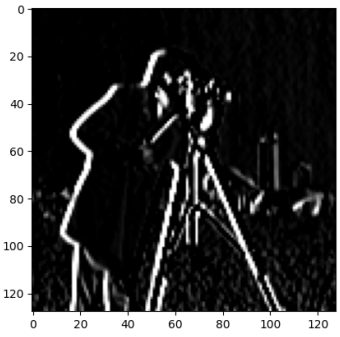
(c) Outline Detector



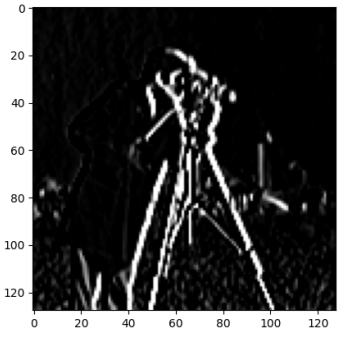
(d) Top Sobel



(e) Bottom Sobel



(f) Left Sobel



(g) Right Sobel

Figure 5: Other Images obtained from FPGA by changing Convolution kernel

1. Slice Logic

Site Type	Used	Fixed	Available	Util%
Slice LUTs	1231	0	20800	5.92
LUT as Logic	1231	0	20800	5.92
LUT as Memory	0	0	9600	0.00
Slice Registers	480	0	41600	1.15
Register as Flip Flop	480	0	41600	1.15
Register as Latch	0	0	41600	0.00
F7 Muxes	8	0	16300	0.05
F8 Muxes	0	0	8150	0.00

(a) LUT Utilization

3. Memory

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	12.5	0	50	25.00
RAMB36/FIFO*	12	0	50	24.00
RAMB36E1 only	12			
RAMB18	1	0	100	1.00
RAMB18E1 only	1			

(b) Memory (BRAM) Usage

Figure 6: FPGA Implemented Design

7 Errors and Shortcomings

The final implementation until 22nd april has a few shortcomings:-

- There is an error in the last pixel that is carried forward through all the stages of the pipeline thereby producing error in the bottom right pixels of the image (as the convolution kernel would use the bottom right pixel 4 times)
- Hence the NMSE of the finally obtained enhanced image from the FPGA compared to the image obtained by implementing the enhancement pipeline in python comes out to be of the order 10^{-7}.

8 Limitations

The final implementation until 22nd april has the following limitations:-

- Although the convolution is parameterised and can be extended to any NxN image and PxP kernel, the padder is only able to pad an NxN image to (N+2)x(N+2) and not account for the size of the kernel.
- The filter values can only be any signed number from -7 to 7. This is because the filter is accepting [3:0] with the MSB acting as the sign bit. However, this can be easily fixed by changing the size of the register suitably.

9 References

- Image Kernels Explained Visually
- Kernel (Image Processing) - Wikipedia
- Signed Number Arithmetic in Verilog
- UART Communication on Basys 3, FPGA Dev Board Powered by Xilinx Artix 7 Part I
- UART Communication on Basys 3, FPGA Dev Board Powered by Xilinx Artix 7 Part II
- Shubhayu Das - UART RX and TX cores
- 15 Part 1: UART-TxD Serial Communication using an FPGA Board — Verilog Step-by-Step Instructions
- 22 Part 2: UART-RxD Serial Communication using an FPGA Board Step-by-Step Instructions