Grayscale Image Processing

Cabir Yavuz, Tejas Sevak
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: cyavuz@oakland.edu, tjsevak@oakland.edu

***Abstract*—In this project, we made a small image processing application on Zybo board. We read RGB bit stream binary file from SD card, did grayscale image processing in embedded system and put the processed image back to SD card.**

I. INTRODUCTION

Nowadays, image processing applications with FPGAs are quite popular because of its parallel computing capabilities. They can do multiple operations at the same time. Image processing requires a lot of computation therefore FPGA's true parallelism makes it great choice for hardware acceleration.

Programming System (ARM processor) is capable of doing this image processing but one of the important points of this project is hardware acceleration. Also, FPGAs are reconfigurable during run time. This gives us the ability to change the bit stream, in other words, operation being done in hardware, in runtime.

Main idea of this project is to put everything we mentioned above together. We planned to use FPGA for image processing. Also, we wanted to create multiple versions of the Gray filter in SD card and load them using Dynamic Partial Reconfiguration techniques we learned in this class.

II. METHODOLOGY

We used 24bit BMP files for this project. In this format, each Red, Green and Blue color is stored in 8bits. This means that their value can be in between 0-255. Originally, we wanted to put BMP file directly to SD card and extract the RGB bit stream in embedded system. However, we couldn't make it work for some resolutions. Therefore, we decided to write our own Matlab scripts to generate bit stream file.

Matlabs imread function read several different image formats and converts into MxNx3 matrix. M and N stands for width and height of the image. After we convert this image into bit stream we load it to SD card.

Our programming system loads this image from SD card. Since we don't know the real size of the image, we have to create dynamic memory. We used the biggest memory portion available in Zynq PSOC which is DDR. It has 512MB capacity.

In our software system, we also included necessary drivers in order to read/write SD card. Then we loaded image in to dynamic memory successfully. We used AXI full interface in order to pass data to hardware system. Because hardware system does the actual process.

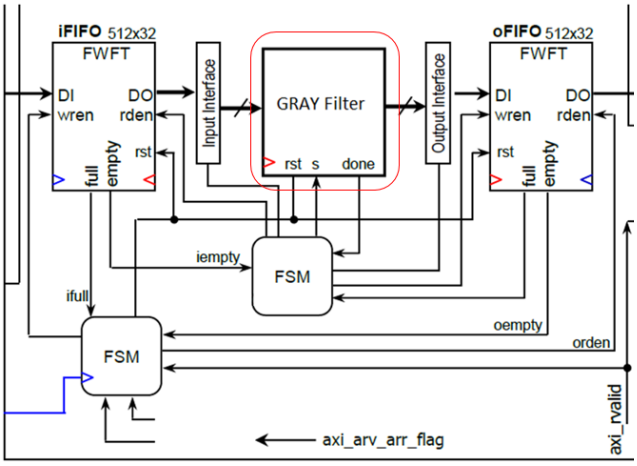
Since we don't know the exact size of the image, we should be prepared for big sized images. This means that we might have to process data more than AXI FIFO is capable of. Therefore, in our software part, we divided input stream data in to small pieces and processed part by part. For example, if we have a 100x100 image, this means that we need process input bit stream data size is going to be around 30KB. This is way more than FIFOs in FPGA can handle. So, we first read first 100 bytes, send it to FPGA, get it back after processing and store them in another dynamic memory. We do this in a loop, so after processing small pieces we combine them together and write in to SD card. This is one of the most important points of software section.

After writing processed data into SD card, we read it via Matlab scripts we wrote in to Matlab environment. In Matlab we construct MxN size matrix from output bit stream. If you realized output size is 1/3 of the input file because grayscale filter converts RGB (24bit) values into Gray (8bit). Grayscale filter is basically an average of RGB values in different weights. After constructing final image, we show the result with imshow Matlab function.

Originally, we wanted to create multiple versions of grayscale filter with different weights so that we would see more light or dark images based on different filter. We thought planned to store these different bit streams in SD card and change our filter in hardware system from programming system using Dynamic Partial Reconfiguration techniques we learned in this class. But time wasn't enough to do that. Thus, we only developed one filter. Now we will look closer to the different part of our system

A. General Block Diagram of Hardware System

You can see the general block diagram of our hardware system below. As you can see, we used the template we used in this class. We used AXI Full interface in order to exchange data between hardware and software system. There are basically two FIFOs in order to handle input and outputs. One of the Finite State Machine in the diagram controls these FIFOs. Other one control input and output to the Gray Filter module. Template we used were giving 2x32bit data to the processing module. But we needed only one 32bit because we can store all Red, Green, Blue values (24bit) in one 32bit. Other information is hardcoded in Gray Filter module because original idea was doing Dynamic Partial Reconfiguration.



B. *Grayscale Filter Explanation*

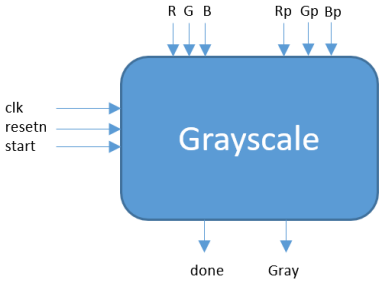
There are different ways of implementing Grayscale filter but maybe one of the most used method is this formula;

$$Gray\ Value = \frac{(R\ * \%X) + (G * \%Y) + (B * \%Z)}{100}$$

Where X + Y + Z = 100. This formula basically takes the average of RGB values with different weights. The reason why percentages can be different is because of human nature. Our eyes are more sensitive to Green light than Red and more sensitive to Red then blue. So, when we compute gray value, we have to take this principle into account. There is so many different possibilities but of course there is a standard for this ratio. ITU-R BT.601is a standard for this ratio.

$$0.299 * R + 0.587 * G + 0.114 * B$$

According to this standard, this is the best ratio of converting RGB to Gray based on human nature.

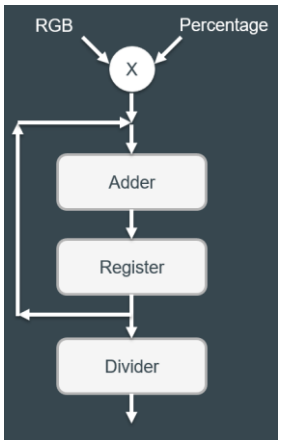


As you can see from the diagram, Grayscale module takes 8 bit R, G, B values, 7 bits Rp, Gp, Bp values. Rp, Gp and Bp are percentage values. These values will be between 0-100 therefore we only need 7bits to represent.

C. *Grayscale Hardware Implementation*

In order to implement Grayscale filter need multiplexer, divider and adder circuits. Since we need to do multiplication for each R, G, B values we could use three different multipliers but this solution would be waste of resources. Hence, we decided to use only one with iteration.

There is a multiplexer that selects each R, G, B and Rp, Gp, Bp values accordingly. In every iteration state machine takes first R and Rp then G and Gp then B and Bp. The result of this multiplication is 15 bit since inputs are 8 and 7bits.



We used Professor Llamocca's multiplier and divider circuits. There were three versions of these circuits; combinatorial, iterative and pipelined. We decided to use fully combinatorial.

As you can see from diagram above, we first multiply the values and add them with previous one. If this is the first run, second operand will be zero because register output will be zero. We store this value in a register.

After we are done with multiplication, we divide the result by 100 because we multiplied by 100 at the beginning. We need to convert it back to 0-255 resolution. This process is just for one pixel, we do that for every pixel.

III. Experimental Setup

We used Xilinx Vivado IDE and SDK tools for this project. Harware part was developed in Vivado IDE and software part was developed in SDK package. We used Digilent Zybo board for implementation. Only additional part we used is an SD card.

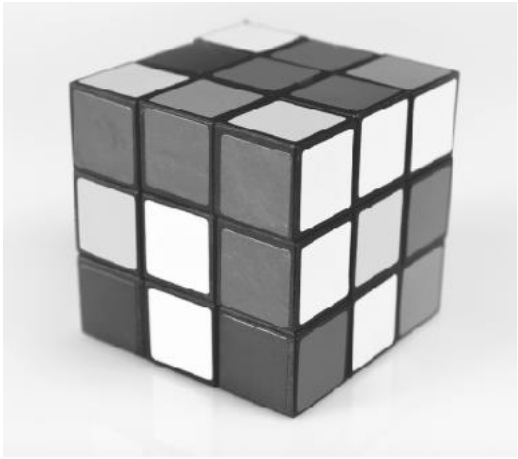
We used Matlab in order to generate input bit stream. We copied the file to SD card from our PCs. We out the SD card to board. We programmed the FPGA first. After that we connected to COM in order to see logging information. Then we programmed the Programming system. After that, we copied output image from SD card to PC. We again ran the

matlab script in order to generate image. We showed the final result in Matlab.

IV. Results

You can see the input image below. This is a 365x419 image. It has different types of colors thus it's suitable for our testing.

After we ran our algorithm, you can see the output below;



Conclusions

As you can see above, we can successfully convert colored image into gray image. However, we couldn't finish Dynamic Partial Reconfiguration part. If we could finish, we were going to generate different variations of gray images, some darker and lighter.

We also couldn't finish reading directly from BMP file. We were successfully extract RGB bit stream from BMP but when we write it to the file, in some resolutions, Windows said that file is corrupted. We didn't want to spend more time on it since we didn't have much time.

In this project and class, we learned how to design an embedded system including both hardware and software using Xilinx tools. We are able to design hardware systems and control them with C code from software. We also learned how to do DPR but we were not able to show it.

References

[1] Llmocca, Daniel. *Reconfigurable Computing Research Laboratory*. < http://www.secs.oakland.edu/~llamocca/arithcores.html>.

[2] Seven Grayscale Conversions.., <http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/>.