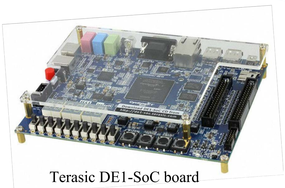
FPGA-Based Image Processing System

Meng Cao, Margaret Luya Gao, Becky Shen, Rainie Yuqing Zhu
Picker Engineering Program, Smith College, MA

INTRODUCTION

The goal of this project is to design an FPGA-based image processing and classification system to investigate the viability of using FPGA as a solution to vet the large volume of user-uploaded content at Facebook. This project aims to develop the Field Programmable Gate Array (FPGA) as a platform to process images. A top-down approach was taken to implementing image processing algorithms on the FPGA board. The functionality of the selected algorithms was validated in higher level programming language (Matlab) and then in C. The algorithm, histogram equalization, was implemented in Verilog.



Terasic DE1-SoC board

IMAGE PROCESSING ALGORITHM

Histogram equalization is a useful technique for enhancing image contrast. It was the first image processing algorithm we implemented in Verilog. We first calculated the histogram of the original image:

$$h[l] = n_l, l = 0, \dots, 255^1$$

Then we normalized the histogram:

$$\bar{h}[l] = \frac{n_l}{n}, l = 0, \dots, 255^1$$

By computing the running sum, we obtained the cumulative histogram:

$$\bar{c}[l] = \sum_{k=0}^{l} \bar{h}[k],\ l = 0, \dots, 255^1$$

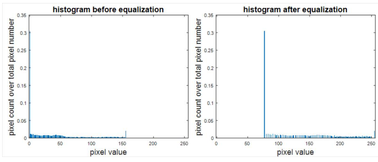
To obtain a cumulative histogram with a constant slope, we adjusted the value of each pixel based on the current cumulative histogram. This resulted in a more equalized histogram with the pixels distributed more evenly across all possible intensity values:

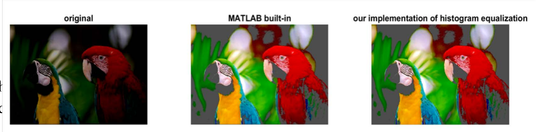
$$l'(x,y) \leftarrow Round(255 \cdot \bar{c}[I(x,y)])^1$$

To avoid drastic distort in the color balance of the image, we only applied histogram equalization to the color channel of the image. We first converted the RGB image to the HSV format:

Maximum: $M = \max(R,G,B)$
Minimum: $m = \min(R,G,B)$
Chroma: $C = M - m$

$$H = \begin{cases} undefined, if\ C == 0 \\ \left(\frac{R-G}{C} + 4\right) \cdot 60, if\ M == B \\ \left(\frac{B-R}{C} + 2\right) \cdot 60, if\ M == G \\ \left(\frac{G-B}{C}\ mod\ 6\right) \cdot 60, if\ M == R \end{cases}^2$$





After the value channel of the image is processed, we converted the image back to the RGB format:

$When\ 0 \le H < 360, 0 \le S \le 1, \& 0 \le V \le 1:$

$C = V \times S^3$

$X = C \times \left(1 - \left|\left(\frac{H}{60}\right) mod 2 - 1\right|\right)^3$

$m = V - C^3$

$$(R',G',B') = \begin{cases} (C,X,0), 0 \le H < 60 \\ (X,C,0), 60 \le H < 120 \\ (0,C,X), 120 \le H < 180 \\ (0,X,C), 180 < H < 240 \\ (X,0,C), 240 \le H < 300 \\ (C,0,X), 300 \le H < 360 \end{cases}^3$$

$(R,G,B) = ((R'+m) \times 255, (G'+m) \times 255, (B'+m) \times 255)^3$

MEMORY DESIGN PROCESS

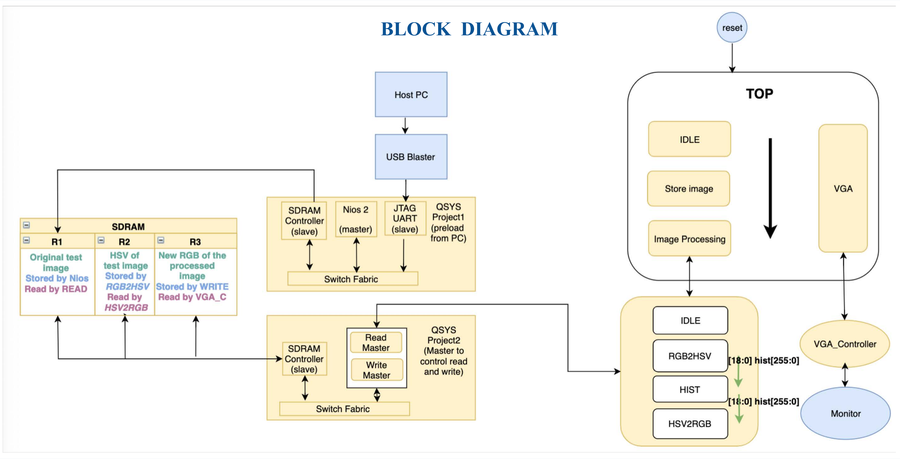
The memory device used for this project is the external SDRAM on the DE1-SoC board. Communication with the SDRAM is required for two parts: the first one is to send the image data from a host PC to the SDRAM, and the second part is to allow the main Verilog program to read and write the SDRAM.

As shown in the block diagram below, two projects were built using Qsys, an Altera system integration tool. In the first project, a Nios 2 processor was integrated as the Avalon Memory Mapped master; an SDRAM controller and a JTAG UART module were integrated as the Avalon slaves. In the second project, a read master and a write master was configured using the Avalon Memory Mapped Master template. The slave was the SDRAM controller. The read and write masters communicate with the SDRAM controller to achieve reading and writing the SDRAM.

SYSTEM ARCHITECTURE

The first component is the image acquisition. A NIOS processor was used to pre-load an image from a computer to the location R1 in SDRAM on the DE1-SoC board. The next component of the project is the image processing and display. The READ master took the RGB data stored in R1 pixel by pixel to the image processing block for image processing. In the first state, RGB2HSV, the RGB value was converted to HSV value and written into R2 in the SDRAM through the WRITE master. The image was converted to an image histogram based on the HSV value of individual pixels on the image, as storing an entire image on the FPGA on-chip memory was expensive in terms of both time and space. In the next state, HIST, the histogram was converted to a cumulative histogram. In the last state, HSV2RGB, the READ master read the HSV data stored in R2 pixel by pixel. The HSV value was redistributed using the cumulative histogram as a weight and converted to RGB value. The WRITE master then wrote the RGB to R3 in SDRAM, where the VGA controller constantly read from at a frame rate of 60 Hz during the image processing process.

BLOCK DIAGRAM



FUTURE STEPS

Our next step is to generate block memory for FPGA and run the algorithm in real-time with a connected camera. We will also implement other image processing and classification algorithm, such as edge detection, segmentation and image classification.

REFERENCES

1. Birchfield, Stan. *Image Processing and Analysis*. Cengage Learning, 2018.
2. "RGB to HSV Color Conversion." *KWh to Watts (W) Conversion Calculator*, www.rapidtables.com/convert/color/rgb-to-hsv.html.
3. "HSV to RGB Color Conversion." *KWh to Watts (W) Conversion Calculator*, www.rapidtables.com/convert/color/hsv-to-rgb.html.

ACKNOWLEDGMENT

The team would like to express our sincere gratitude Facebook liaisons, Dr. Ahmad Byagowi, Maria Karstens, and Mike M. Lambeta. We would like to thank Dr. Arman Pouraghily for his technical advise. We appreciate the long-term encouragement from Professor Susannah Howe, our senior capstone project coach. Finally, without the support from Picker Engineering Program and Department of Computer Science of Smith College, the progress of this design project would not be possible.