University of Strathclyde

RI@S End of Internship Report

Software Define System on Chip

Author:
Andrew Maclellan

Supervisor:
Dr. Louise Crockett

September 12, 2016

Contents

Introduction

The project undertaken during the 10 week internship focused on a field of electronic engineering involing embedded systems on FPGAs (Field Programmable Gated Arrays). A lead company in the field of FPGA design, Xilinx, released a new embedded FPGA device, MPSoC (Multi Processor System on Chip), along side a new software program SDSoC (Software Defined System on Chip). This new device allows software engineers to program the System on Chip devices using C/C++ only where previously it would have required a hardware engineer with good knowledge of FPGA design. This new program opens many doors in System on Chip design where software engineers with limited knowledge of the hardware design can program these devices using familiar programming languages such as C/C++.

The purpose of the internship is to discover how to program operates, the do's and dont's, and use the program to produce an example of image processing application.

Content Section

The System on Chip devices used were from a family of Zynq FPGA devices. The Zynq is an ARM-based processor that runs with an FPGA.

Learning Material

In order to be more familiar with Zynq and System on Chip design (SoC), the first 2 weeks of the internship involved reading and completing tutorials from the 'Zynq Book' (written by the university), as well as attending a two day training course at the Rutherford Appleton Laboratory for the SDSoC program. The Rutherford Appleton Laboratory is a research facility in the south of England that focuses on many areas of research in various different fields.

Research

Once the theory of Zynq SoC devices and the basics of SDSoC were understood, the next section of the internship involved going through the process of how a software engineer would typically create an application using SDSoC as well as identifying any areas that could cause issues.

Hardware Platform

The most fundamental part of the SDSoC program and how it operates is through its 'hardware platform'. The 'hardware platform' is a predefined hardware build indicating to the program what areas of the FPGA the application will be using (GPIO, HDMI, VGA, ethernet, etc...). For a software engineer to be able to use SDSoC, they will need to have a premade platform created by a hardware engineer for their application.

To further understand the significance of the hardware platform, a simple hardware design was created to target the GPIO (General Purpose Input Output) ports so that the board can 'talk' to the outside world throught LEDs, switches, and/or buttons. This proved to be more than trivial as it invloved extensive research on the concept of hardware design to understand when to use certain parts of the board. For example, when to add in clocks for timing. It was also especially difficult synqing the hardware platform that was created with SDSoC to 'tell' the program how many clocks were in the platform, alternate ports for better performance, and making the platform into a readable format for SDSoC to understand. This process is yet to be automated meaning that the user must manually create all the files needed to make SDSoC understand the platform.

To conclude, it is highly recommended to use a premade platform created by a hardware engineer in order to ensure a properly working project.

Image Processing Application

For this image processing application a premade platform provided by Digilient for the Zybo (the manufacturer of the Zybo Zynq SoC board) was used instead of creating a platform from scratch. This ensured that the platform was correct and working and any issues that would arise would be due to the software design and not the hardware platform.

The application will operate by streaming in video from the laptop display through HDMI, passing through all the pixels individually through the Zybo while manipulating them, and outputting to the 1920x1080 monitor through VGA.

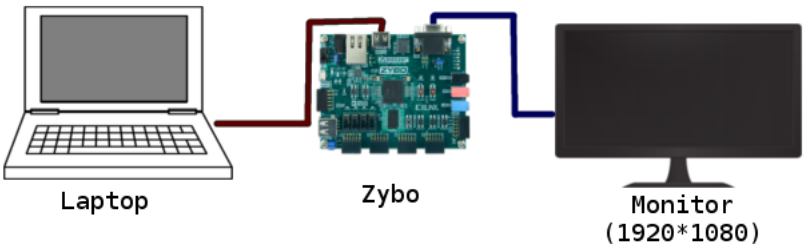

Figure 1: Image Processing Setup

The video stream coming from the laptop was manipulated in 4 different ways using 4 filters. A passthrough filter, a grayscale filter, an average filter, and a sobel filter. How these filters operate will be explained later. All 4 filters operate in a similar way where the video stream is feeding one frame at a time into the Zybo, each pixel is read in one at a time from each frame and then the filter is applied to every pixel.

The most important aspect of this application is that any code that is written for the hardware needs to be synthesisable. This means that the C/C++ code must be written in a way where it is possible to convert it into hardware, meaning that there cannot be any code that does not translate into hardware such as infinitely incrementing loops, unknown variable sizes, etc...

Passthrough Filter The passthrough function is fundamental to understanding how the pixels are read in and manipulated. The function operates by reading in each pixel of a frame and separating the red, blue, and green values and then recombining the red, blue, and green values back into a pixel and feeding it into the output frame into the monitor.

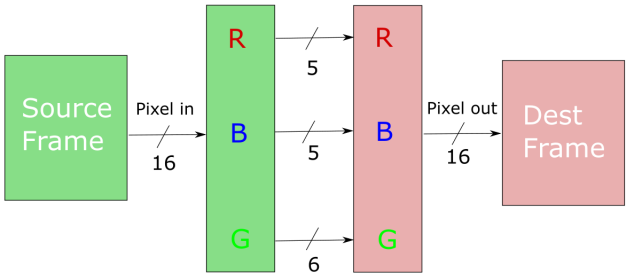

Figure 2: Passthrough Filter

To take this one step further, when recombining the RBG colours, the red and blue component of the pixel can be swapped and recombined. This makes anything that was previously red blue and anything that was blue red.

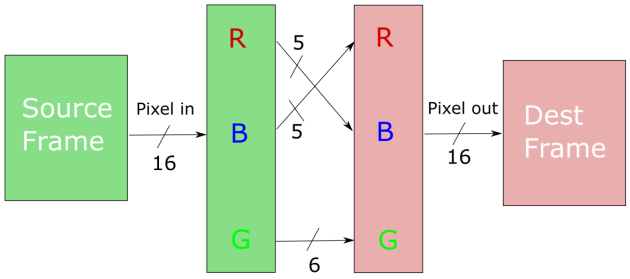

Figure 3: Colour Swap

When the passthrough colour swap filter is live and running on the Zybo, without hardware acceleration being specified, there is a delay in the video stream output and the frames that appear on screen every second greatly reduce (usually 60 frames per second FPS). This is because the filter is initially run within the processor (the Zynq chip) or the 'brain' of the Zybo and not the FPGA. In the processor every calculation is done one after another, however in the hardware of the FPGA calculations can be done in parallel and in more efficient ways such as pipeling and unrolling. Pipeling and unrolling are techniques unique in hardware where they make calculations and function more efficient at the expense of more resources from the FPGA. In order to increase the output frame rate, SDSoC gives the user the option to hardware accelerate the filter within the FPGA as long as it has been programmed to be synthesisable. Once it is hardware accelerated, the frame rate greatly increases and the filter runs smoother compared to it being on the processor.

Grayscale Filter The grayscale function is very similar to the passthrough function where the pixels are read in and the RBG colours separated. However with the grayscale function, red, blue, and green are processed mathematically to determine the corresponding gradient of gray for each pixel.

$$Y = R * 0.3 + B * 0.113 + G * 0.587$$

The above equation weights each RBG colour differently when finding the average gradient due to the human eye perceiving each of the colours differently. For example, green

is perceived more vividly than red and blue.

Applying the same concept of reading in the pixel and separating the colours.

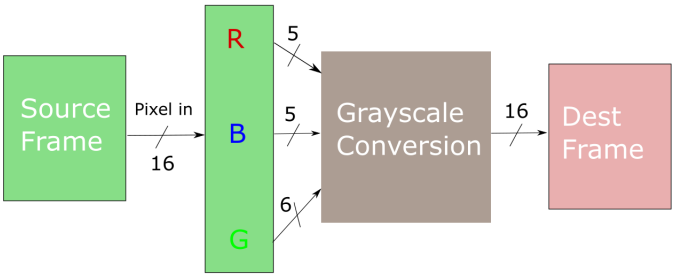

Figure 4: Grayscale Filter

The same concept applies for the grayscale filter when the filter is run on the processor compared to the hardware. The filter performs better on the hardware because the average gradient calculation can be completed fast using techniques such as pipeling and unroll which can be specified within SDSoC.

Average Filter The third filter operates by converting the image firstly to grayscale and then taking the average of the target pixel and any pixel surrounding it. This operation is completed by using a 'window' to pass over each frame of the video stream that comes into the Zybo.

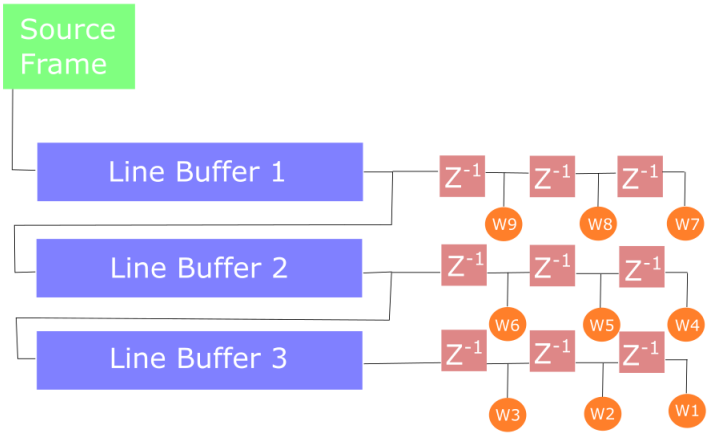

Figure 5: Average Filter

The image above may seem confusing at first, but after a short description it will

become clearer. The average filter operates on 3 rows at a time hence the 'line buffers'. The target pixel sits in the centre of the 9 boxes and the other 8 are the pixels surrouding it. The 3x3 window passes over the full frame and calculates the averages of all 9 pixels for every new target pixel. This achieves a blur effect to the video stream, making everything seem out of focus.

The benefit of using the FPGA hardware can clearly be seen with this filter because of the increase in arithmetic operations. The filter is calculating the average of multiple pixels within a frame and 60 frames pass through the Zybo a second. Running this function on the processor only proves to be very slow and unreliable.

Sobel Filter The final filter created was the Sobel filter named after Irwin Sobel who developed the method. The function uses the concept of a 'window' to pass over the frame the same as the average filter. The difference here is instead of taking the average of all surrounding pixels, the pixels are weighted depending on the change in gradient of the grayscale image.

The Sobel filter uses two windows. One window for finding the change in gradient in the x-direction i.e from left to right of the image and the other window for finding the change in gradient in the y-direction. The two windows look like this:

1	2	1	-1	0	-1
0	0	0	-2	0	-2
-1	-2	-1	1	0	1

Figure 6: Sobel filter windows

By using these windows and the method of passing them over the frame, the output video stream will be a black and white gradient image where the lighter the gradient (white) the sharper the gradient on the image. For example, all edges in the image will be white because of the sudden gradient change.

Figure 7: Sobel example

(The above picture was taken from a reference online due to there being no way to capture the monitor output of the Zybo.)

Conclusion

This summer's internship proved very beneficially to my course and taught me various skills that I can use in my further studies such as programming a System on Chip device, write basic image processing functions, work in a research environment with other interns and PhD students, and keep to deadlines set by the team. The internship also taught me how to properly research a topic and how to document my progress and findings properly. All in all, this summer's internship was very fun and has me thinking of progressing through a research route after I graduate.

References

[1] The Zynq Book - Bob Stewart, Louise Crockett.

[2] Digital Image Processing - Rafael C. Gonzalez.

[3] Design for Embedded Image Processing on FPGA's - David Graeme Bailey.