Ambient Backlight

Adam Sunderman and Jeramie Vens

# Introduction

This project was to create an ambient backlight controller for a media system.  The idea is to take the colors at the edge of the screen and extend them onto the wall behind the display to give the illusion of a larger display and to ease the transition on the viewer.  Studies have shown that this can help reduce the strain on the viewer and reduce headaches when watching TV in a dark room.  This is a passthrough system where the input to the TV is connected first to the Zedboard, and then the output from the Zedboard is connected to the TV or other display all via HDMI.

# Hardware Implementation

The hardware was composed of two key parts (shown in figure 1). The first part was the image pipeline.  The pipeline took in video data from the HDMI and converted to AXIS format.  That was then run through the VDMA to allow the software access to the video data.  The VDMA was configured to both read and write to the same frame buffer therefore giving the full 60Hz refresh rate we needed.  The output of the VDMA was converted back to HDMI and sent out the port.  The other part of the hardware was the PWM IP core.  We wrote the core ourselves modeling it off of the ATMEL PWM modules with a clock prescaler, a top register, and a duty cycle register.  Using these registers we were able to control the intensity of the red, green, and blue light to make any color we needed.

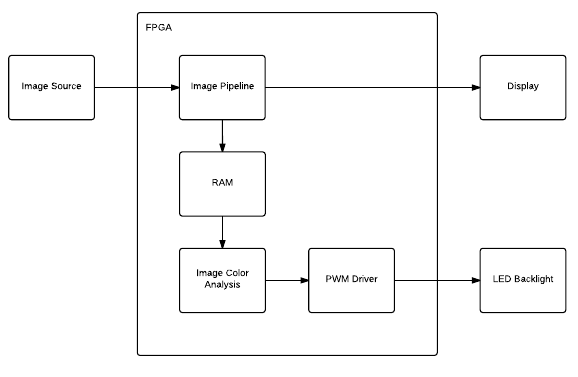


Figure - Hardware Implementation

# Software Implementation

Our software initialized our hardware pipeline using code from the FMC documentation. After initializing, our software read the current frame from the frame buffer for processing. The primary responsibilities of our software loop included: converting the yCrCb data to RGB data, averaging the pixels around each LED location on the screen, and sending the RGB values to the LED’s via our PWM IP core. We developed a PWM API to allow our main processing loop to easily read and write the important registers of our PWM IP core.

The software loop made use of a helper function that averaged the RGB value of a rectangular group of pixels after receiving an x and y location as well as the width and height of the rectangle of pixels to be averaged. These functions returned the average RGB value in a rgb\_color structure that was then passed to the PWM API to set that specific LED.

In addition to this base functionality, we enabled our user to control the ambient depth. The ambient depth is defined as the distance (in pixels) from the edge of the screen that are taken into account when averaging the RGB values. We created a mode of operation that allowed the user to change the ambient depth by pressing the up and down buttons. This mode was activated by turning SW0 on. In this mode, our program also gave the user feedback by drawing white lines on the screen to represent the grid of pixels taken into account by each LED. This mode did slow down the video playback as we had to write to the output buffer instead of just reading it. In this mode, the delay of image processing and the arithmetic to place the ambient depth lines is noticeable, however, it quickly bounces back to the desired 60 fps after you have set your desired ambient depth.

# Conclusion

Overall, we were able to meet all of our baseline goals and managed to meet many of our stretch goals. This was accomplished because we were able to start our work early and were able to split the work that needed to be done into logical sub-sections. Through our combined effort we were able to get the hardware pipeline up and running rather quickly. From there we designed the requirements for the software and PWM IP core and split the work accordingly. Having a well defined method of interfacing (via the PWM API) before hand, our individual work was able to connect and work quickly and efficiently.