Project 1: Color Wheel Implementation

Nishad Saraf

Portland State University, Oregon

ECE 544: Embedded System Design with FPGA

February 3, 2017

Table of Contents

Introduction

This project is based on the Digilent Nexys4 DDR board which incorporates the latest Artix Series 7 Field Programmable Gate Array(FPGA) from Xilinx. Apart from the onboard switches, LEDs, pushbuttons, RGB LEDs and seven segment display this project uses two additional peripherals, Digilent's 96x64 Pmod OLEDrgb display and Pmod ENC rotary encoder.

From the point of view of a user, this project this project displays a specific color on the OLED display and RGB LEDs. The color displayed depends upon the values of hue, saturation, and value feed through the rotary encoder, left-right and up-down pushbuttons respectively. As per the selection of detection method of PWM to calculate duty cycle, duty cycle of signal arriving on RGB LEDs is displayed on the two 4-digit onboard seven display. Duty cycle corresponding to red, green, and blue color content in original signal shown. The user selects the method of detection through the onboard slide switch SW0. High on SW0 implies that PWM detection is done using a specialized hardware while a low SW0 means that the PWM detection is done in software.

Hardware Setup

Nexys4DDR board's expansion ports JA and JD (bottom row) are configured such that peripherals Pmod OLEDrgb display and PmodENC rotary encoder can be connected. Board can receive power from the Digilent USB-JTAG port (J6) or from an external power supply. An external battery pack can be used by connecting the battery's positive terminal to the center pin of JP3 and the negative terminal to the pin labeled J12, directly below JP3. Since the main regulator on the Nexys4 DDR cannot accommodate input voltages over 5.5VDC, an external battery pack must be limited to 5.5VDC. An external power supply delivering 4.5V to 5.5V and at least 1A of current can be used by plugging into to the power jack (JP3) and setting jumper

J13 to “wall”. Although as the power supply required by both the peripheral is equal to 3.3V, the board can efficiently function on the power drawn from the micro-USB port.

PWM signal arriving on the pins of RGB LEDs is passed onto the input pin of GPIO0. This connection is done to later extract the PWM signal in software detection approach. Three additional GPIO IPs are added to the design which is created based on the “Getting Started in ECE 544” document. Each of these GPIO module is dedicated for handling one single color - Red, Green, or Blue and is configured with two 32-bit output channels.

Apart from two clock signals clk_out1 and clk_out2, for hardware detection of PWM one extra clock signal pwm_clk at clk_out3 port is added to the Clocking Wizard that operates at 10MHz. This clock is specifically used as an input to the PWM detection module.

Software Setup

Now for the hardware detection approach new source file containing program for PWM detection written in Verilog is added to the file hierarchy in Vivado. After the above-mentioned modifications, subsequent changes are made to n4fpga.v file which includes:

1. New port definitions of GPIO and clock.
2. assigning RGB signal to the input channel of GPIO0.
3. PWM detection module is instantiated thrice one for each color component.

PWM Detection

Hardware approach

The inputs to the PWM detection module are the clock, reset and the signal (W_RGB1_color) from RGB LED, while the outputs are the count values when the signal W_RGB1_color goes high and low. To store these values two 32-bit reg type variable H_Cntr and L_Cntr are defined.

Now on positive edge of the clock pulse or on the occurrence of negative edge of reset (happen when system restarts) reading the input signal is initiated. When the signal goes high temporary counter register which keeps a track of high counts is increment by 1. Similarly, when the signal goes low another temporary register that keeps track of low count is incremented by 1. But what about the next PWM cycle? How can we possibly know that one complete input cycle is finished and we need to reset the counters? To solve this issue, we initialize two register flags H_Flag and L_Flag. When the signal is high H_Flag is set to 1 subsequently when the signal goes low L_Flag is also assigned value 1. At this point both the flag has value 1 which means we have our entire positive and negative part of the cycle. Now if the signal goes high and as the flag are set indicated that the next input cycle has begun and we need to output the final counts stored in the temporary counter register to H_Cntr and L_Cntr, clear the flags to default value (zero) and reassign value zero to the temperoary low counter. As 1 count in high signal is used to load the temporary value on the output line and resetting the counter and flag the temporary high counter must be initialized to 1 instead of 0.

With the help of the `XGpio_DiscreteRead()` function we can easily read the values arriving on the each of the 3 GPIO ports. Arithmetic operations are done in C to display the value of duty cycle on seven segment display.

Software approach

Through the hardware setup explained above we can extract the signals arriving on red, green, and blue pins of RGB LEDs. But this is not enough to read the values on this port the CPU must be interrupted after equal intervals to time to update the current count value. The Fixed Interval Timer (FIT) is clocked at 100MHz and set to raise an

interrupt signal after every 2500 clock pulses (i.e. **after every 25 microseconds CPU is interrupted**). This interrupt is INT0 which is the highest priority interrupt. The frequency (40KHz) at which the GPIO0 is checked for updated value is 10 time higher than the frequency of signal (4KHz) arriving on the RGB LED pins.

Whenever the CPU is interrupted a predefined function called as FIT_Handler() is invoked. It is with the help this function the counting of high and low pulses for each of the color component is done. In the n4fpga.v file each color component is bundled in one single signal as:

```
gpio_in = {5'b00000, w_RGB1_Red, w_RGB1_Blue, w_RGB1_Green};
```

gpio_in is 8-bit wide channel by properly masking other bits it possible to read one color component after the other. Rest of the programming logic remains the same as the one written for PWM hardware detection in Verilog.

Minor Optimizations (after demo)

Following shortcoming in the project as pointed by the teaching assistant Mr. Chetan Bornarkar are rectified as follows:

1. Continuously reading the switch value: Earlier SW0 was designed to read input from the user only in the initial 2seconds phase. Later this was modified by executing the switch read logic in while infinite loop defined in the function ColorWheel(). Now the user can switch to the other method of detection at any given time.
2. Hue value calibration: Map function was added in the design to wrap around the hue value 0 to 359 and from 359 to 0.

3. Incorrect duty cycle shown on the display due the unwrapped hue value: With the correct calibrations of hue value this issue was resolved.

Challenges Faced

Memory segmentation fault

On Vivado v2016.2, the system gave "Memory Segmentation fault – Failed to load hardware design" when an attempt was made to export the design to the SDK. The reason behind this issue was the incompatibility of Vivado v2014.2 to run on Windows 10 home OS. After downgrading to Vivado 2015.4 this issue was solved and I was able to export the design to SDK.

Corrupted .c and .h files

After successfully programming the FPGA, OLED and rotary encoder didn't function as expected, while other onboard peripherals were functioning as they should be. The non-functioning of OLED and rotary encoder was due to one of the corrupted file among PmodOLEDrgb.c, PmodOLEDrgb.h, PmodENC.c, and pmodENC.h. Reloading the new files resoled the problem.

No signal on gpio_in

Reading the gpio_in port returned 0 value because the RGB signals were not wrapped inside the gpio_in as it was mentioned in the project 1 handout. Bundling the signal red, green, and blue signals desired signal was extracted in software.

References

1. Digilent Nexys4 DDR™ Board Reference Manual. Copyright Digilent, Inc.
2. Digilent PmodCLP™ Parallel LCD Reference Manual. Copyright Digilent, Inc.
3. Digilent PmodENC™ Reference Manual. Copyright Digilent, Inc.
4. Getting Started in ECE 544 (Vivado/Nexys4) by Chetan Bornarkar.
5. AXI GPIO product guide by Xilinx
6. Fixed Interval Timer product guide by Xilinx