Lab 2: VGA Display and Light Sensor Interface

ECE 3829

Nicholas Lanotte & Matthew DiPlacido

September 15, 2017

Table Of Contents

Introduction

The Basys3 FPGA is equipped with a variety of I/O connectivity, including four seven segment displays, a VGA out and many I/O pins to connect external devices to the board. In this lab, the application of these ports is explored and combined into a system capable of reading an SPI light sensor, driving a 640x480 pixel display based on input from two switches (with one state driving the display based off of light sensor data), and displaying the light sensor value on the seven segment display.

Problem

The first problem was to modify the simple seven-segment display module to support not only characters 0-9 and A-F on one anode but on all for anodes. Implementing such a feature meant a clock needed to be added so all 4 digits can be viewed at the same time.

Next the VGA display modules needed to be created using the supplied VHDL file along with a 25MHz clock from an MMCM. Using the MMCM the FPGA needs to output to a 640x480 screen resolution through the VGA port to an external display. Based on the switch inputs this screen needs to be able to output a completely yellow screen, a screen with alternating red and green rows 16 pixels thick, a black screen with a white 64x64 block centered in the screen, and then a black screen with the same block centered in the middle but with the color depending on the light sensor reading.

An SPI interface to communicate with the light sensor was needed. It has to use a 1MHz clock to read in the light sensor data 10 times per second. A shift register needs to be utilized to read and store the data and the light sensor value needs to be displayed to the seven segment display twice, ranging from 00 to FF.

Lastly, bonus features could be added to show an in depth understanding of this lab.

Overview of Design Approach

In order to implement such a module in Verilog several other modules are needed to simplify the top module and to make the debugging process much simpler. Figure 1 shows the overall implementation of the top module for this lab.

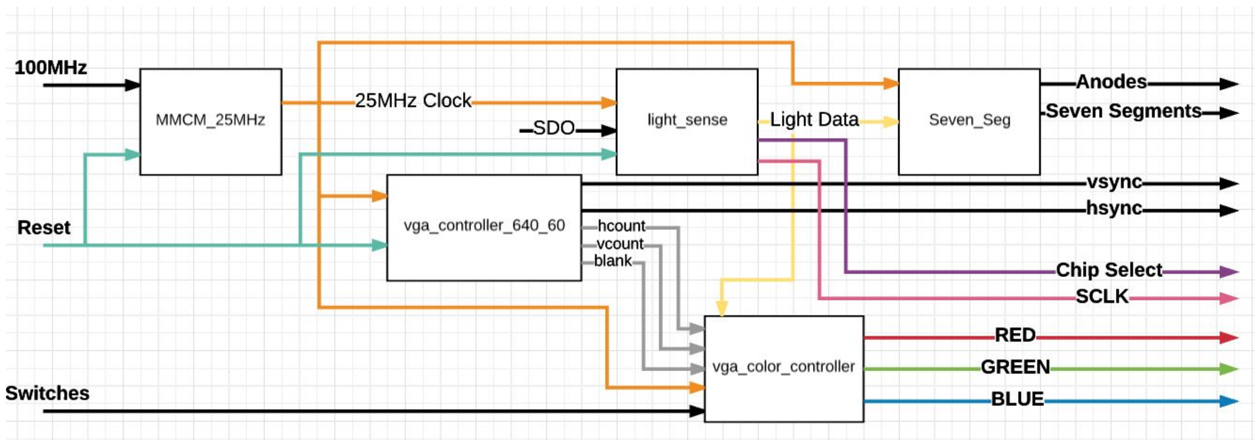


Figure 1: Design Approach Block Diagram

As shown in figure 1 there are 5 additional modules that needed to be created in order to create the top module for this lab; seven_seg, light_sense, vga_color_controller and the 25MHz MMCM. The first module completed was the seven segment display module because the majority of that module was completed in the first lab. The seven_seg module was modified so that it would toggle through all four anodes faster than the human eye can see so that it appears as though all four seven segment digits are always on continuously. The seven segment display takes inputs from a 16-bit input bus (which was originally the input from the switches) as well as an input clock signal to drive the anodes, and the module outputs directly to the seven segment displays and to the anodes.

After completion of the seven segment display the addition of the vga controller module was completed using the supplied code from Digilent. The module was created as a VHDL file and the code was pasted with in and the file was saved and then added to the top module.

The next step was implementing the color controller module. This module drives the display and implements the necessary requirements for part 1 of this lab. The color controller module should display a completely yellow screen when all of the switches are low, display alternating red and green bars of height 16 when the first switch is high, display a solid white block 64x64 pixels in the center of the screen when a second switch is high and the first is low, and lastly display another block in the center of the screen of the same size as the previous state however the color of the block should depend on the value of the light sensor. Creating this module started with setting up the needed inputs and outputs which are depicted in Figure 1. An input clock, switches input, a vertical pixel count indicating the current vertical pixel location, a horizontal pixel count indicating the current horizontal pixel location, a blank signal indicating if the current pixel location is on or off screen, and the light sensor data were added to the module as inputs. The outputs from the module are three 4-bit buses that drive 12 pins on the VGA port, one bus for Red, one for blue and one for green.

The next module necessary needs to drive the light sensor data input to the color controller module and handle the communication between the light sensor and the board. This module, called light_sense, powers the clock signal to the light sensor, the chip select pin on the light sensor, and reads the data from the serial data out pin on the light sensor, all of which is driven by the 25MHz clock and the reset button.

The majority of the light sensor module is used to define the SPI communication between the board and the light module. Figure 2 below shows the communication between the board and the light sensor as implemented in this lab. In order to transfer one complete light sensor reading, the chip select needs to be driven low for one clock cycle and after 3 serial clock cycles 8 bits are passed over SDO during the positive edge of the clock cycle and then 4 trailing zeros on the positive edge of the clock are output, this means 16 clock cycles on SCLK are needed to completely take one light value from the sensor. SCLK must be between 1 and 4MHz for the device to function properly so, for this lab, a 1MHz clock enable flag was created using the input 25MHz clock. In figure 2 the Chip Select is indicated by Channel 1 (yellow), the serial data is shown by Channel 2 (teal) and the clock signal is shown by Channel 3 (purple).

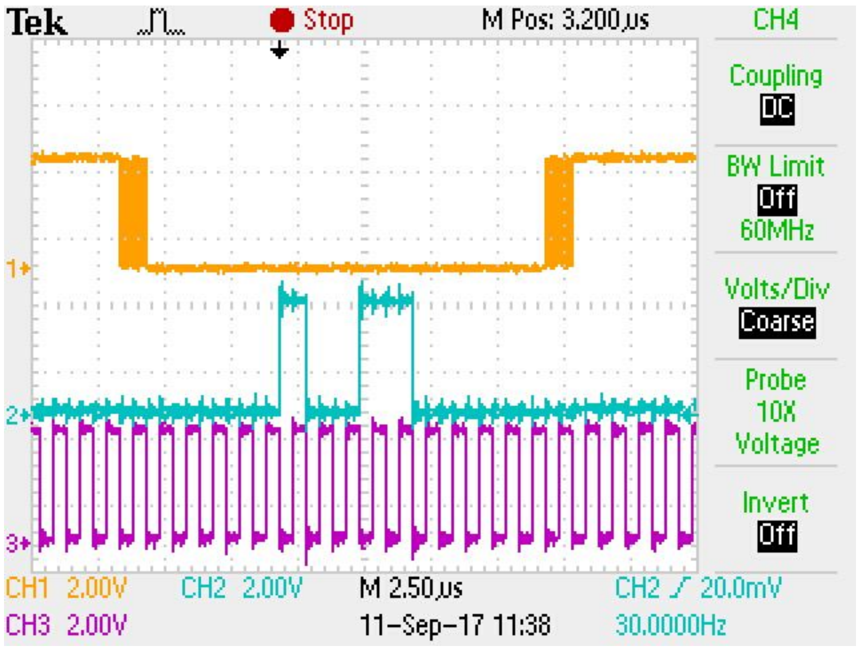


Figure 2: Oscilloscope reading of the CS, SCLK and SDO pins for the light sensor

In order to record this information from the light sensor so that it can be used in the rest of the module, the 8-bits of light data need to be recorded using a shift register. SDO is only 1 pin on the board and it is driven by the clock so when Chip Select is low the bits of information can be shifted into the register and once 16 clock cycles of the 1MHz clock have completed the register is done being filled and the data can be used.

Lastly the clock driving all of these modules needed to be created before the top module could be completed. An MMCM was created using the Clocking Wizard in Vivado. This module consisted of the input 100MHz crystal oscillator located on the board and an output clock of 25MHz that drives all of the other modules mentioned above.

Once all of these modules were defined, the top module was constructed using all 5 modules and several wires to pass information from one module to another. The top module consisted of four inputs; the 100MHz clock, a reset button, the SDO from the light sensor and the switches needed to control the vga color module. The top module then handled all of the outputs from the various modules; three 4-bit color buses, SCK, Chip Select, Anodes, seven segment values, vertical synchronization for the VGA and horizontal synchronization for the VGA display.

This lab also offered bonus points for interesting features added to the existing modules. The feature chosen added another state for the VGA output that created rows of pixels that would “fall” down the screen. The color of these pixels was determined by the light sensor. 15 eight-bit registers were added to the color module to store the last 15 readings from the light sensor, each being used to determine the color of a 32 pixel high row on the screen. With each new reading the old readings are shifted down one register, shown by the colors shifting down screen, creating the “falling” effect seen. The colors can range from blue to green to purple depending on the reading from the light sensor.

Final Solution

The FPGA used 207 Flip-Flops (0.50% of the available) for the final design. This number is only 87 if excluding the flip-flops used for the extra credit portion of this lab. The full summary can be viewed in Figure 4 of the appendix.

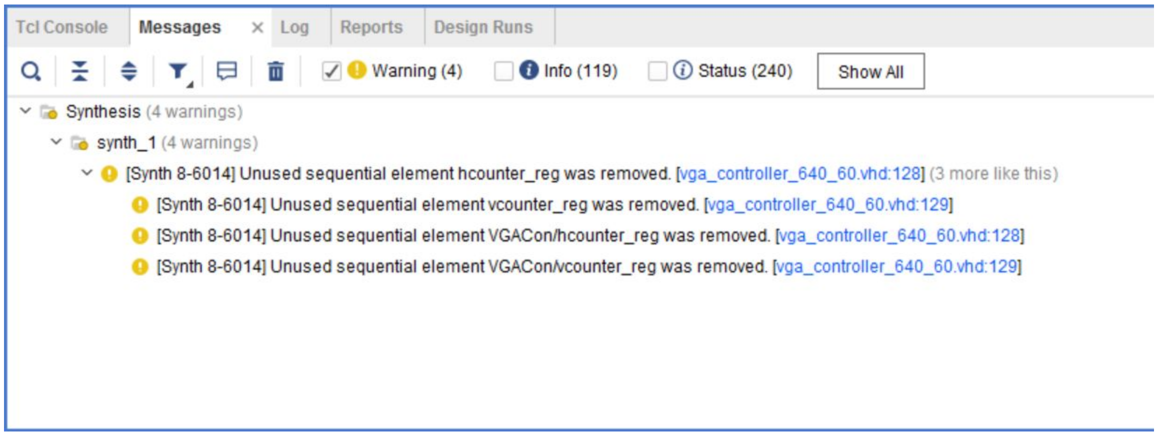
24 of the flip flops are used in the “VGA controller_top” module, which is the top module for this design. The 1MHz clock signal output used as SCLK for the light sensor is of type register because a flip flop is needed to be able to toggle this output high and low to create a 1MHz square wave. The Chip select output is also of type register so it can be toggled high and low at a rate of 10Hz. To create these slower clock signals the input clock from the MMCM was used in a counter, which would create an enable signal used to toggle the SCLK and CS signals. To bring the 25MHz MMCM clock to 1MHz, the counter needed to count to 25, requiring 5 flip flops to store this value. To create the 10Hz clock, the 1MHz clock was used to trigger a counter, counting to 100,000,requiring 17 flip flops to store.

The color module used 132 flip flops, but of those 120 were used in the extra credit portion. The 12 flips flops required for this lab were three 4 wire busses used to output the intensities of the Red, Green and Blue sub pixels. The 120 flip flops used for the extra credit portion are used to store data. The extra credit feature needs to store the last 15 readings from the light sensor, each being 8 bits, resulting in 120 flip flops necessary to hold all of this data.

The “Light_Sensor_Controller” Module uses 20 registers total. This module is designed to be used by the top module to output an 8 bit signal that updates 10 times per second. 8 flip flops were used to store this output signal so that is stays constant until being updated. There also was a 12 bit shift register used to read in the data, which required the remaining 12 flip flops. The Light sensor sends 15 bits of data per readings, 3 first 3 bits being zeros, the next 8 bits being the data, and the final 4 bits are more zeros. This module reads in the data line and shifts the data over but because the 3 MSB of the data are zeros they are not needed. This allowed the SPI shift register to be optimized to 12 bits, which would delete the first 3 bits of data from the SPI controller as it shifts the rest of the data.

The “Seven_seg_full” module uses a total of 25 flip flops. 4 flip flops are used to store the 4 bit data that determines, which seven segment displays to turn on. Another 4 flip flops are used to store the 4 bit data that is sent to the “display” module, which determines the 7 bit output signal to the seven segment displays. The final 17 flip flops are used to count to 100,000 in order to divide the input clock by 100,000. A clock is needed to alternate which seven-segment display is on so dividing the input clock will slow down the refresh rate of the screen. In this implementation each display is refreshed 62.5 times per second. This module uses the "Display" module to determine the output signals to the seven segment displays. The "Display" module uses 7 flip flops to store set the 7 bit output to the seven segment displays.

The sum of the above flip flops is 208, which is one more than the summary states. The totals above were calculated by reading the verilog code to adding all of the registers. The warnings did not say any registers were removed from the verilog files so some of those register type variables must have been made into a different component. The remaining flip flops that make up the total must come from the VDA controller module. The scope of this course does not require the ability to read VHDL but the warnings from synthesis indicate that the hcounter and vcounter registers were removed so it is assumed that this module has few or possibly no flip flops.



```
Synthesis (4 warnings)
  synth_1 (4 warnings)
    [Synth 8-6014] Unused sequential element hcounter_reg was removed. [vga_controller_640_60.vhd:128] (3 more like this)
      [Synth 8-6014] Unused sequential element vcounter_reg was removed. [vga_controller_640_60.vhd:129]
      [Synth 8-6014] Unused sequential element VGACon/hcounter_reg was removed. [vga_controller_640_60.vhd:128]
      [Synth 8-6014] Unused sequential element VGACon/vcounter_reg was removed. [vga_controller_640_60.vhd:129]
```

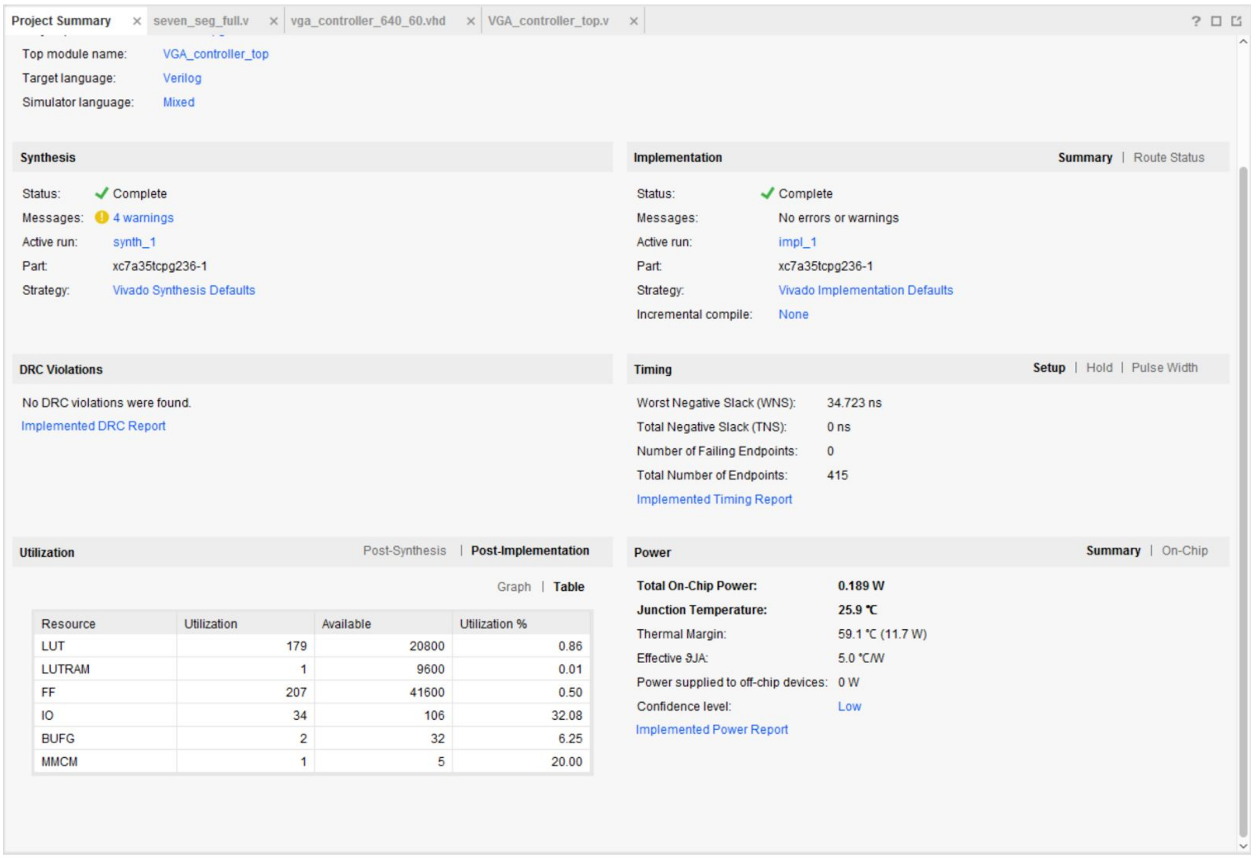
Figure 3: Warning Messages

Figure 2 shows the warning messages from the synthesis of the design code. There are no other warnings or errors from implementation or generating the bitstream. These warnings are from the provided VGA Controller code, which is in VHDL format. VHDL is not in the scope of the course so there was no way to correct these warnings. Fortunately they are only saying that certain registers were not necessary. These registers were for the vertical and horizontal count that determine the pixel being written. The design code for this lab determines the pixel colors based on these values but only in certain ranges so many of the bits from the vertical and horizontal counters are not needed. The registers corresponding to these bits are the ones being removed from the design, creating these warnings.

Conclusions

In this lab a Basys 3 board was used to implement a VGA display controller that displays different images on the screen based on input from two switches, a reset button and from a light sensor module. The device was capable of displaying a completely yellow screen, a screen with alternating red and green bars of height 16, a black screen with a white box 64x64 pixels centered on the screen and a black screen with a box 64x64 pixels centered on the screen with its color depending on the light sensor readings. Multiple modules were created to complete specific tasks and to make debugging the process easier and to make understanding the operation of the top module easier. Lastly an additional feature was created that showed the last 30 colors and cascaded them down the screen to show changing values over time, and to demonstrate a more complete understanding of Verilog. The main problems encountered in implementation involved the creation of slower clocks. The clocks were first created using logic gates, which created a lot of warnings and errors during implementation and generating the bit stream. After reviewing the class notes an understanding on how to properly create clocks that are synced was obtained and implemented into the design. There were other smaller warnings that came up during the creation of the modules but they were solved by reading the warnings and changing the code based on the warning suggestions. With a lot of trial and error, the system was created properly meeting all requirements. The main lessons learned were how to create a proper slow clock using clock enable signals and syncing them with the faster 25MHz clock. It is also much more clear when to use an always statement as opposed to an assign statement, as well as when reg type components are necessary. This project could be improved further by modifying the VGA_Controller code to be better optimized. The seven-seg module was created to be generic for use in later labs, but if this were to be implemented in a system it could be better optimized to use the other clocks from the top module rather than using the 25MHz clock and creating a slower one internally.

Appendices


Project Summary | seven_seg_full.v | vga_controller_640_60.vhd | VGA_controller_top.v

Top module name: VGA_controller_top
Target language: Verilog
Simulator language: Mixed

Synthesis

Status: ✓ Complete
Messages: 4 warnings
Active run: synth_1
Part: xc7a35tcpg236-1
Strategy: Vivado Synthesis Defaults

Implementation — Summary | Route Status

Status: ✓ Complete
Messages: No errors or warnings
Active run: impl_1
Part: xc7a35tcpg236-1
Strategy: Vivado Implementation Defaults
Incremental compile: None

DRC Violations

No DRC violations were found.
Implemented DRC Report

Timing — Setup | Hold | Pulse Width

Worst Negative Slack (WNS): 34.723 ns
Total Negative Slack (TNS): 0 ns
Number of Failing Endpoints: 0
Total Number of Endpoints: 415
Implemented Timing Report

Utilization — Post-Synthesis | Post-Implementation — Graph | Table

Resource	Utilization	Available	Utilization %
LUT	179	20800	0.86
LUTRAM	1	9600	0.01
FF	207	41600	0.50
IO	34	106	32.08
BUFG	2	32	6.25
MMCM	1	5	20.00

Power — Summary | On-Chip

Total On-Chip Power: **0.189 W**
Junction Temperature: **25.9 ℃**
Thermal Margin: 59.1 ℃ (11.7 W)
Effective ϑJA: 5.0 ℃/W
Power supplied to off-chip devices: 0 W
Confidence level: Low
Implemented Power Report

Figure 4: Project Summary

Attached to this report is first the signed off Verilog code from lab. The formatting of this code was not very good so another copy of the code is also attached with better formatting. Both are identical in content but the later code is properly formatted and thus easier to read. Please look at the properly formatted code when evaluating it.