Lab 2: Light Sensor and VGA Monitor Display

Yil Verdeja, ECE Box 349
Yeggi Lee, ECE Box 191
Lab Section A01
September 14, 2018

TABLE OF CONTENTS

1 Introduction

The purpose of this lab was to design and implement several interfaces in order to control a VGA module and light sensor using the Basys3 board. The VGA monitor was programmed to display a series of different patterns including solid colors, stripes, and blocks of certain sizes using switches. For the light sensor, a MMCM was used in order to create a 25 MHz clock required for the sequential logic in the lab. Several counters and a shift register were also created in order to develop an SPI interface able to read 8-bits of light information from the sensor. The 8-bits of data would then be shown in the seven segment display in hexadecimal.

2 Design Description

The figure below shows the block diagram of the *controller* module. The controller is in charge of reducing the internal clock to 25 MHz and instantiating lower-level modules that will produce the output described in the *Introduction*.

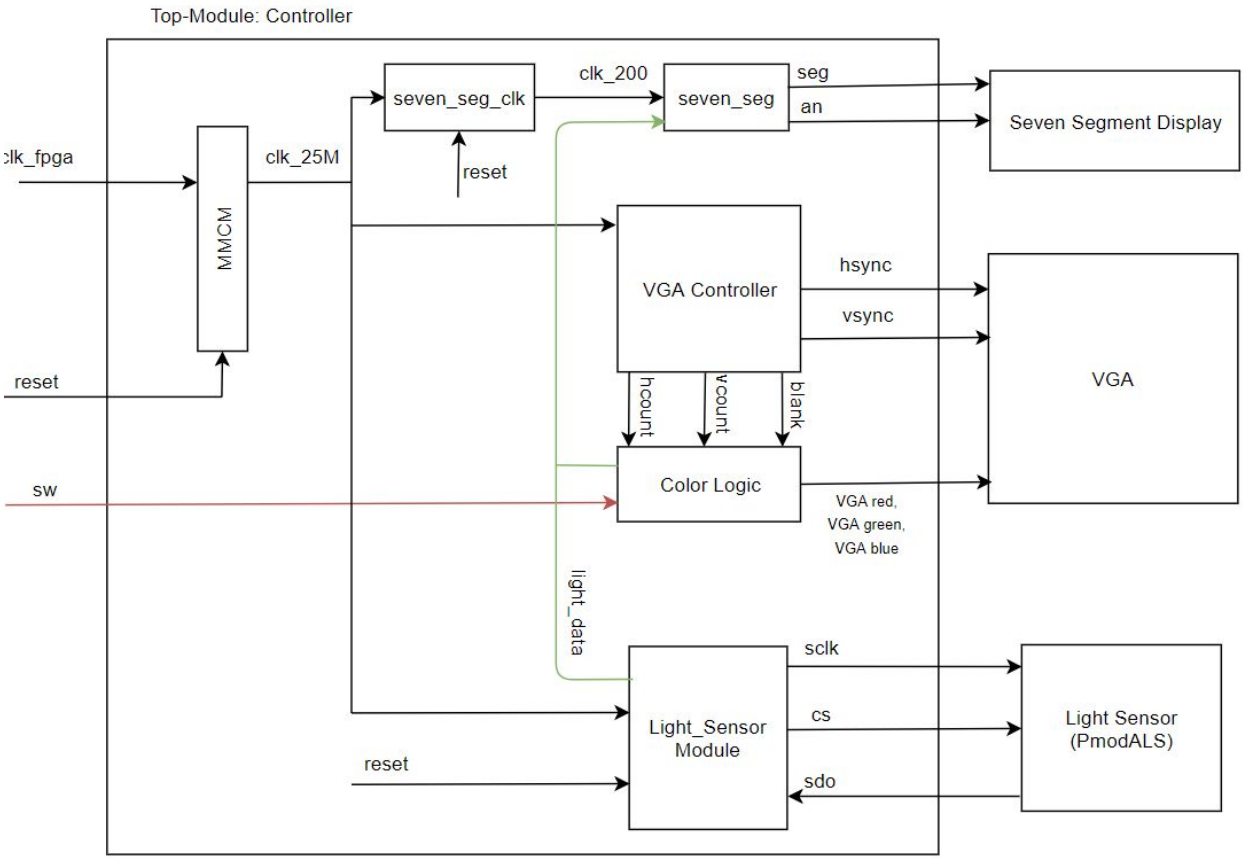


Figure 1. Block Diagram

The Basys 3 board has a 100 Mhz crystal oscillator clock. The MMCM (Mixed Mode Clock Manager) module is used to generate a 25 MHz clock (*clk_25M)* from the crystal oscillator

frequency, which is used throughout the controller to provide sequential logic to other lower-level modules.

The *seven_seg* module uses two seven-segment displays on the Basys 3 board. Since only one seven segment display can be lit at a time, the two anodes that control each display are toggled at a frequency greater than 60 Hz. By increasing the frequency of each anode to 100 Hz, the switching between each anode is undetectable to most humans as it produces a steady light to an individual.[1] Since each anode uses a 100 Hz, the seven segment has a 200 Hz input clock that has been slowed down from the 25MHz clock.

The *light_sensor* module acts as the SPI interface and is in charge of creating both the 1MHz serial clock and the 10Hz chip select for the light sensor. It also collects the correct 8-bit data from the light_sensor by using a 16-bit shift-register.

The VGA logic is created through a *color_logic* module and a provided *VGA Controller* module. The *VGA controller* contains the logic to generate the synchronization signals, horizontal and vertical pixel counters and video disable signal for the 640x480@60Hz resolution. On the other hand, the *color_logic* module uses the horizontal and vertical pixel counters, and a combination of switches to color the screen with the given patterns:

1. Completely red display,
2. Vertical bars of alternating yellow and green colors with each vertical bar 32 pixels wide,
3. A black screen with a white block 32 pixels wide by 16 pixels high in the top left corner of the screen,
4. A black screen with a blue block 64 pixels wide by 32 pixels high in the bottom right corner of the screen
5. A display that changes colors from red to white using the collected light sensor data

2.1 Light Sensor Interface

In this section, a light sensor interface is implemented using a serial clock, a chip select, and a shift-register to collect the data from the PmodALS light sensor.

From the lab specifications, it is required to generate an ADC serial clock (SCLK) at 1MHz with a 50% duty cycle. To accomplish this task, a counter that counts to 25 (0 - 24) divides the 25MHz clock to the desired frequency. A clock enable signal, which is used to create the chip select frequency, is set to active high once the counter reaches its max count (i.e 24).

[1] https://skeptics.stackexchange.com/questions/3348/can-the-human-eye-distinguish-frame-rates-above-60-hz

As specified, a new light sensor value has to be captured every 100 ms (10Hz). From the PmodALS reference manual, the light sensor delivers a single reading in 16 SCLK clock cycles by bringing the CS pin low.[2] Thus, since SCLK is 1 MHz (1 µs), the CS pin should stay low for 16 µs. By using a similar procedure to slowing down a clock, a counter from 0 to 99,999 is used to lower the 1 MHz SCLK to 10 Hz. When the counter is less than 16, the pin is placed on active low. Once the counter is above or equal to 16, it's placed on active high and is thus not collecting any information from the light sensor.

Once both the SCLK and CS pins are working, then the light sensor can output data once CS is set to active low. For 16 µs, it should collect 16 bits of information. From the ADC081S021 data sheet as shown in the figure below, the 8-bits of light sensor information can be found knowing that the first 3 leading bits will always be zeros.

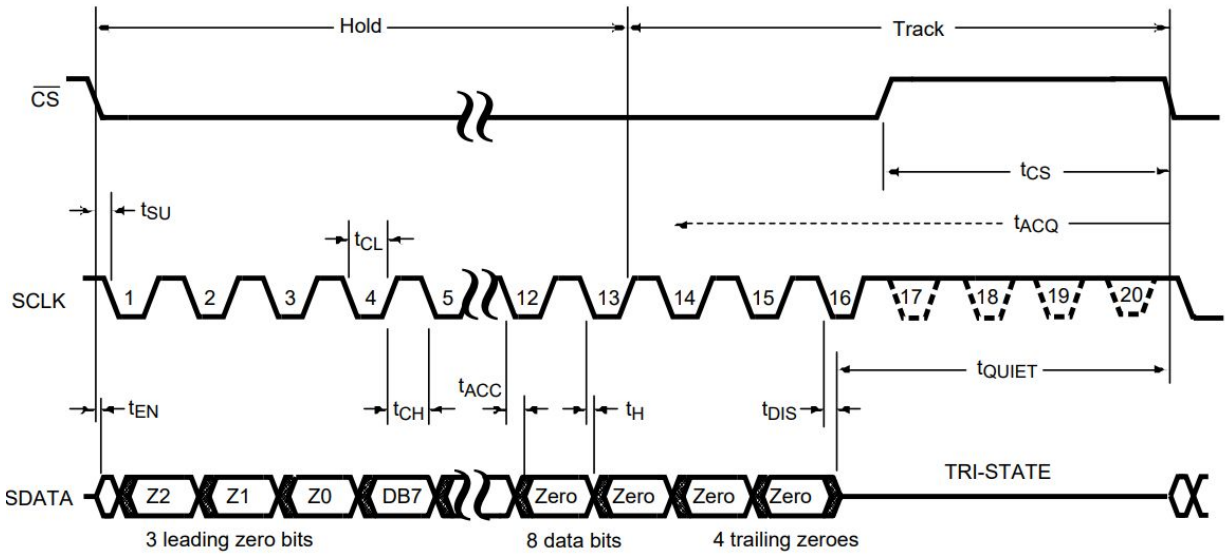


Figure 2. Serial Timing Diagram of the ADC081S021[3]

Using an oscilloscope, it was verified that the SCLK and CS pins were working as expected due to the way the SDO pin reacted as light was manipulated on to the sensor. The oscilloscope readings are shown in Figure 3. As the light was changed, so were the bits coming from the data output. From no light to being very lit, the 8-bit data from the light sensor ranged from 0 to 255 respectively.

[2] https://reference.digilentinc.com/reference/pmod/pmodals/reference-manual
[3] http://www.ti.com/lit/ds/symlink/adc081s021.pdf

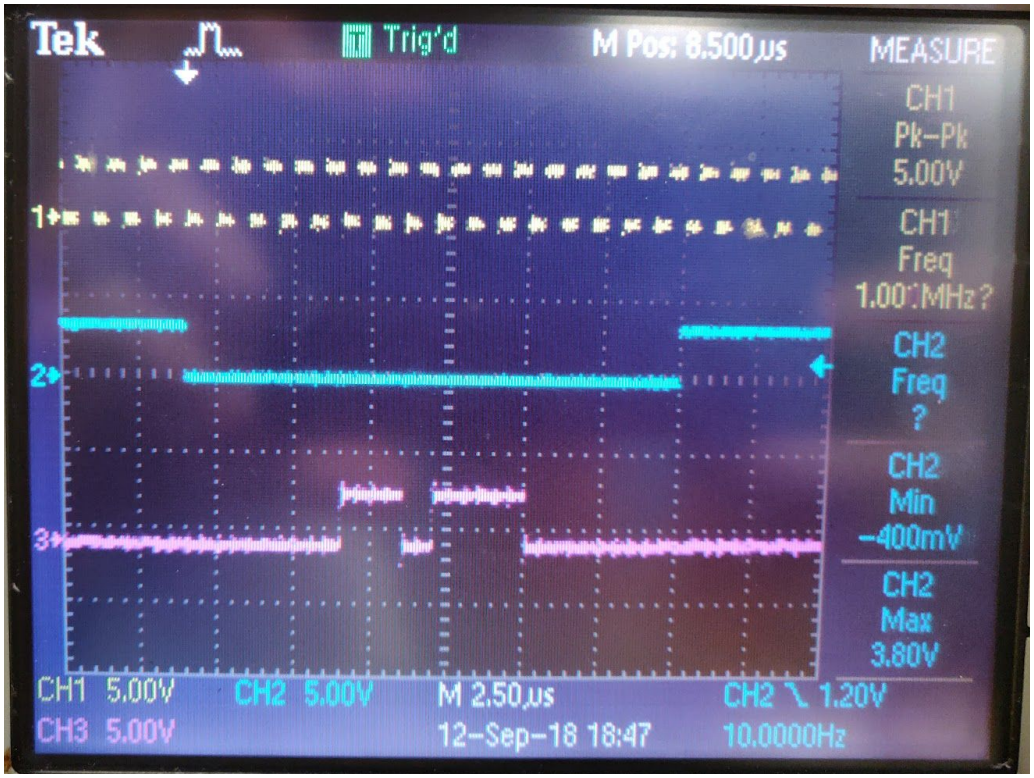


Figure 3. SCLK (CH1), CS (CH2), and SDO (CH3)

Finally to collect the data from the light sensor and output it on the two seven-segment displays, a left shift register was implemented. Everytime the chip select would be low and SCLK was enabled, then the data of a 16-bit register was shifted to the left, where the 1-bit from SDO would be placed in the LSB (Least Significant Bit). After 16 SCLK clock cycles, 8-bits of data from bits 12 to 5 are outputted to the seven-segment display.

2.2 VGA Display

In this part of the lab, a classic 640x480 VGA display was created. A regular VGA is mainly composed of five pins for red, green, blue, vertical sync and horizontal sync. The horizontal count (*hcount*) indicates the horizontal position of the pixels in relation to the display while the vertical count (*vcoun*t) indicates the vertical position.

As shown in the figure below, the VGA display can be separated into active display and the front and back porches. However, in order to simplify the code, the variable *blank* was used in order to show whether the current position was within the active LCD display.

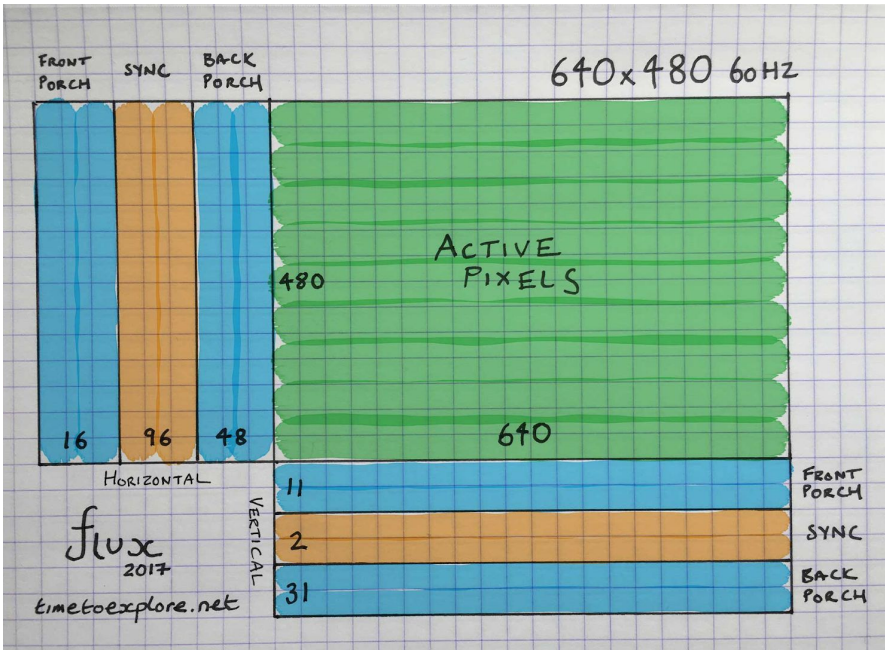


Figure 4. 640 x 480 VGA Display Diagram[4]

The main purpose of the *color_logic* module was to create different patterns and colours on the VGA display through the use of three switches. The input RGB controls the colour shown depending on the value. Each separate colour is composed of 4 bits, making a total of 12 bits. In order to implement the patterns, a case structure was used as the inputs are mutually exclusive.

The case structure compares an expression concatenated from *blank* and *rgb* to a series of values in order to produce the proper pattern. The first design was creating a completely red screen meaning that the *hcount* and the *vcount* were unnecessary. Only the *rgb* was used in this case for the solid colour.

On the other hand, the second case created vertical bars of alternating yellow and green colours with each bar being 32 pixels wide. As shown in the table below, every 32 pixels, the 5th bit alternates value from 1 to 0; therefore, a stripe pattern can be made. When hcount[5] is high, the bar is yellow. Otherwise, the bar is green.

Table 1. Vertical Bars Pattern Table

Decimal	Binary	Colour
32	0010 0000	Yellow
64	0100 0000	Green

[4] https://timetoexplore.net/blog/video-timings-vga-720p-1080p

96	0110 0000	Yellow
128	1000 0000	Green
160	1010 0000	Yellow
192	1100 0000	Green

The next case involved actually using the *hcount* and the *vcount* to create a black screen with a white block 32 pixels wide by 16 pixels high in the top left corner of the screen. Because the block was requested on the top left corner, *hcount* has a value of 0 to 31 while the *vcount* has a value of 0 to 15.

Similarly to the previous case, the last one included creating a black screen with a blue block 64 pixels wide by 32 pixels high in the bottom right corner of the screen. However, because the block was to be on the bottom right corner, 640 pixels was subtracted by 64 making *hcount* > 576. For the *vcount*, 480 pixels was subtracted by 32 resulting in *vcount* > 448.

Finally, for the extra credit, an extra case was added to include the light sensor. Depending on the *light_data* received from the sensor, the display would change into a range of colors varying from red to white. This was done by splitting the light sensor data to change only the green and blue colours, and keeping red high at all times.

3 FPGA Resource Usage and Warnings

This section gives an explanation to the FPGA resource usage and the warning messages received.

3.1 Flip Flops (FF)

As shown in the figure below, the number of flip-flops used after implementation were 94. By looking at all the modules, this value can be determined.

Number of Flip Flops in the ***light_sensor*** module:

Variable	FF
counter_sclk	5
counter_cs	17
data_reading	16
light_data	8
sclk	1
cs	1
Total	**45**

Due to the synthesis warnings *Synth 8-3332* as shown in section 3.2, the number of total flip flops is reduced to 45.

Number of Flip Flops in the ***clk_sevenseg*** module:

Variable	FF
counter	17
clk_200	1
Total	**18**

Number of Flip Flops in the ***seven_seg*** module:

Variable	FF
count	4
Total	**4**

Number of Flip Flops in the ***vga_controller*** module:

Variable	FF
hcounter	11
vcounter	11
HS	1
VS	1
blank	1
Total	**25**

Together, they all add up to 92 flip flops. Looking at Figure 6, our analysis was off by 1 flip flop from the *light_sensor* module. However we were not sure where that was the case.

Resource	Utilization	Available	Utilization %
LUT	100	20800	0.48
LUTRAM	1	9600	0.01
FF	94	41600	0.23
IO	33	106	31.13
BUFG	2	32	6.25
MMCM	1	5	20.00

Figure 5. Post-Implementation FPGA resource usage

Name	Slice LUTs (20800)	Slice Registers (41600)	Bonded IOB (106)	BUFGCTRL (32)	MMCME2_ADV (5)
controller	100	93	33	2	1
clk_fpga_to_25M (clk_...	0	0	0	2	1
ls (light_sensor)	28	46	0	0	0
sevenseg (seven_seg)	11	4	0	0	0
sevensegclock (clk_se...	13	18	0	0	0
vgac (vga_controller_6...	47	25	0	0	0

Figure 6. Flip Flop usage per module after synthesis

3.2 Synthesis Warnings

The two synthesis warnings on *Synth 8-3917* are due to the constant value being used to drive the anodes for the seven-segment displays. This is what was intended so those warnings can be ignored.

The three synthesis warnings on *Synth 8-3332* were also expected since the first 3 leading bits of the 16-bit register are zeros and hence did not need to be there. A possible solution is to have made a 13-bit shift register.

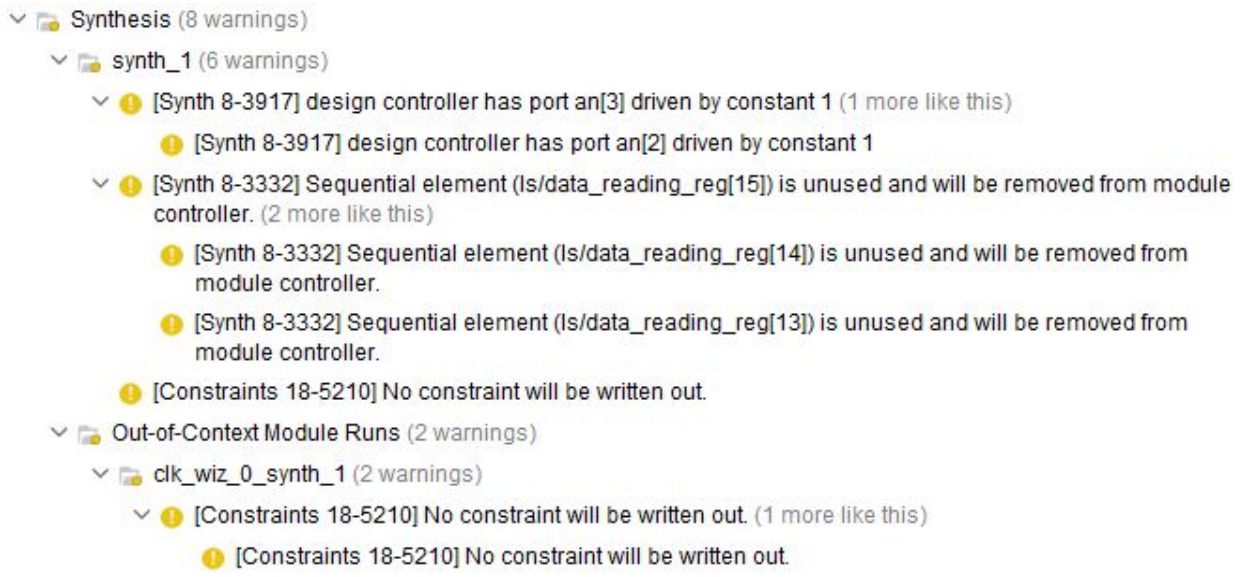


Figure 7. Synthesis Warnings

4 Conclusion

The main problem during the implementation phase was correctly using the proper clocks. Initially, slower clocks were used in the sensitivity list which did not cause problems at first. After lecture, we realized that our design would have created a clock skew which would have impacted the flip-flop timing and caused timing delays. Another issue that occured was that we set all the resets to be asynchronous instead of synchronous. However, the design required the reset to be generated by a set of internal conditions.[5] This would allow the reset to prevent and filter out glitches between clocks.

Through this lab, we were able to see the difference between sequential and combinational logic in actual code and how it affected the outcome. Additionally, we learned how to properly create slower clocks and the fact that the same high-speed clock must be used for all implementations.

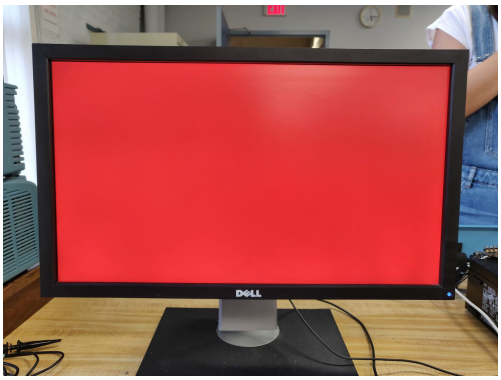
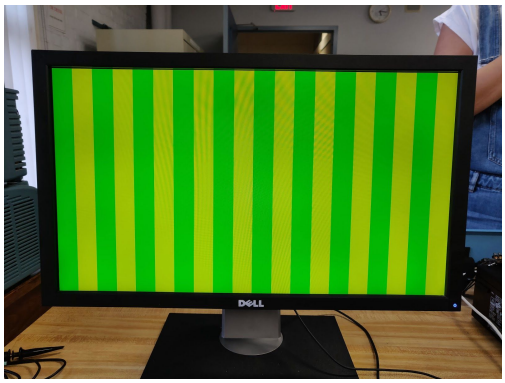
Possible improvements in the code would be to change the counter in *seven_seg* to be 2 bits instead of 4 bits, change the non-blocking assignments in *color_logic* to blocking as it uses combinational logic, and reduced the amount of flip flops in *light_sensor* by setting the 8-bit *light_data* output using a continuous assignment.

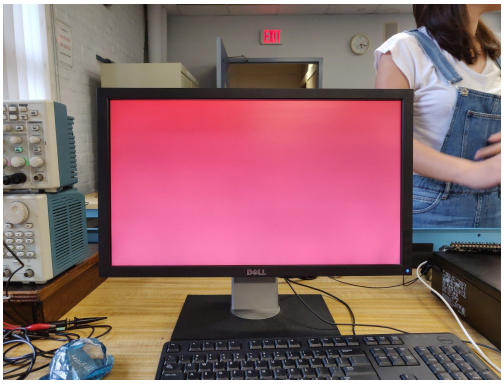
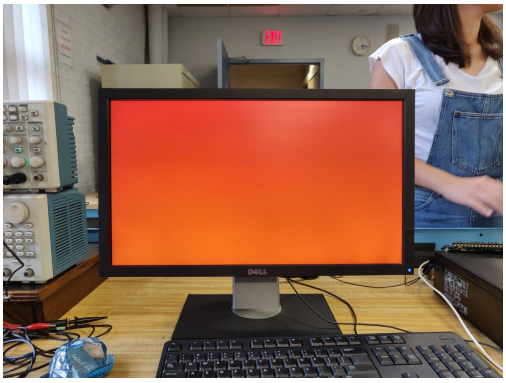
An extension could be to create a moving animation on the VGA display. This lab only introduced the beginnings of what one can do using the FPGA and a VGA monitor. We would like to further explore how to create something more complicated than just displaying different colours on the screen.

[5] https://m.eet.com/media/1121857/chapter2_clocks_resets-03.pdf

5 Appendix

Images produced by the FPGA in the VGA Monitor

[6]

[6] It's interesting how changing the solid colour produces a gradient looking output. This is due to the fact that only individual pixels are set at a time, and so as the light is slowly affected, so are the colours the pixels are displaying.