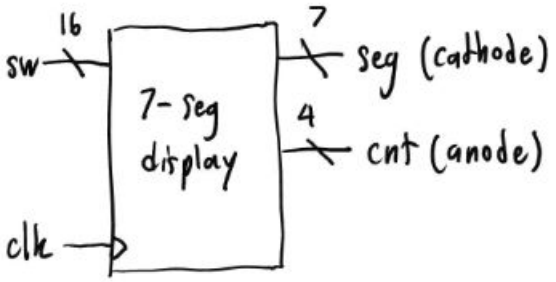
## Introduction

This lab is more focus on sequential logic and many peripheral ports such as VGA and SPI interface. Also, this lab involves using provided modules by Xilinx and Digilent such as mixed mode clock manager (MMCM) and Digilent VGA controller. Unlike previous lab, encountering unwanted results or problems and debugging occur regularly. Regardless, we can accomplish the combined results to show a specific pattern given an input and a value of light sensor correctly including the screen which shows a yellow block whose position depends on the sensor.

## Discussion

Preliminary: Seven segment display

The block diagram contains 2 inputs: 4-bit values for 4 panels combined as 16-bit variable and clock signal. The outputs are signals to use for seven segment display for FPGA board.

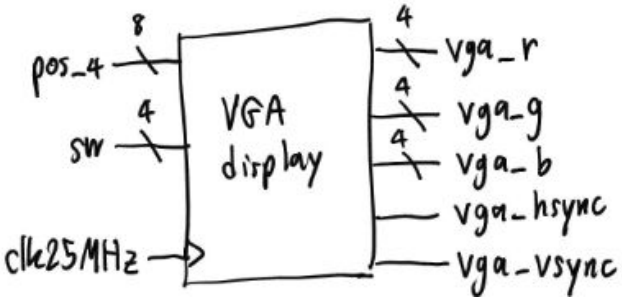



Previously created on the first lab, the seven segment module without clock signal can only displays at specific panel at the time by setting output variable ‘anode.’ In this module, to be able to show all four digits together, the seven segment tries to show each digit so fast that human eyes cannot refresh faster than the changing rate of anode variable, so they see all digits shown at once. Because of the required clocks for other parts, mixed mode clock manager (MMCM) is set to output 25MHz clock from original 100MHz clock on FPGA board. Even 25MHz is too fast for using for each digit; the display cannot clearly illuminate and darken the LEDs correctly. Hence, the counter for clock divider is introduced.

The problem encountered is the speed of clock is too fast to directly use to switch anode. Before noticed, the display almost shows all LEDs with less light (from obviously red light to almost-dead-battery red light). The solution is to decrease the clock; a new counter is introduced to use as trigger instead of 25MHz clock. In this case, 1.25kHz clock is used by counting to 1000. However, the suggested refresh rate on the Diligent manual is 1kHz to 60Hz. In this module, the deflect is not obviously shown; it is not relatively noticeable compared to one before adjustment.

Part 1: VGA display

The block diagram in this part has inputs as integer value (represented by 8-bit variable), 4 switches to select which pattern is shown, and clock signal. The output are for VGA port.



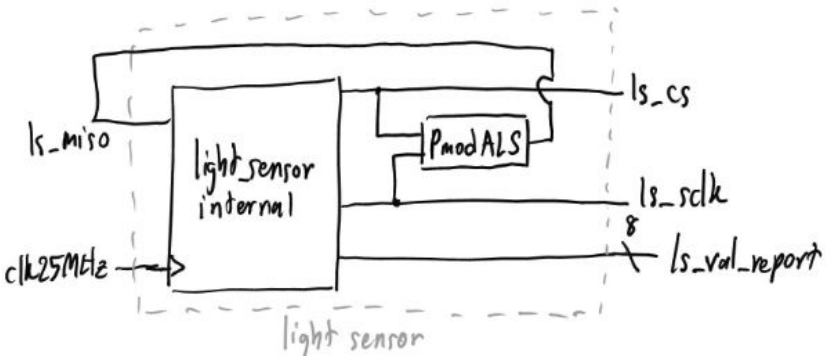

Because the clock was set previously for seven segment module, it is moved to this module, and the input variable in the seven

segment module is changed from realizing 100MHz clock to 25MHz clock, which will be connected in VGA display module. There is the module from Diligent (named 'vga_controller_640_60.vhd'), which already sets vsync and hsync and gives vertical position and horizontal position on the screen as well as the validity of the given position (expressed as 'blank' indicating whether the pixel is on the screen) from provided clock signal. Hence, what is left to cover is to correctly assign values of red-green-blue (RGB) on given pixels. Note that on an invalid pixel (pixel whose blank value is 1) must be explicitly set to black. Next, slide switches is used as the priority encoder, so if the first choice is chosen at the same time as the second choice, the display will show the first choice. For color assignment, 'rgb_to_each' module is instantiated for setting ease. Because separated assigns are harder to deal with, i.e. the need to assign three buses to specific value. Also, it is more well-known to assign RGB at once. The module is straightforward; it handles given RGB value and separate to specified buses for R, G, and B values. Moreover, because of this module, parameters for colors can be easily set such as all constant parameters which is initiated in the module (namely COLOR_*). For position consideration, a screen can be considered as 2 dimensional array of pixels whose first and second indexes indicate the positions of row and column respectively. Finally, the pixels are ready to be filled. First, blank pixels need to be checked first and set color to black. Then, 4 cases are handled separately. In green-filled display case, pixels are directly set to green (from parameter 'COLOR_GREEN'). In eight horizontal color case, it is known that the 'vga_vpos' is for row (or horizon), so color is assigned accordingly. It is known that there are 480 rows of pixels; it is divided to 8 sections, 60 rows for each. Hence, if-elseif expressions can be used to specify which section has which color. Similarly, in yellow block in the middle case, if-else can be used to separate sections to two sections: yellow block and remaining. Last, similar to the third case, the forth gets dynamic horizontal location from an input. A variable (bus) is added to store the start position of yellow block; because it is known that the block has 32 pixels wide, the end position can be simply calculated.

The non-black color set to blank pixels was encountered that initially made supposed-to-be green-filled display to black-filled display. This problem was solved by adding if-else clause to check for blank pixels first and then continue regularly.

Part 2: Light sensor interface

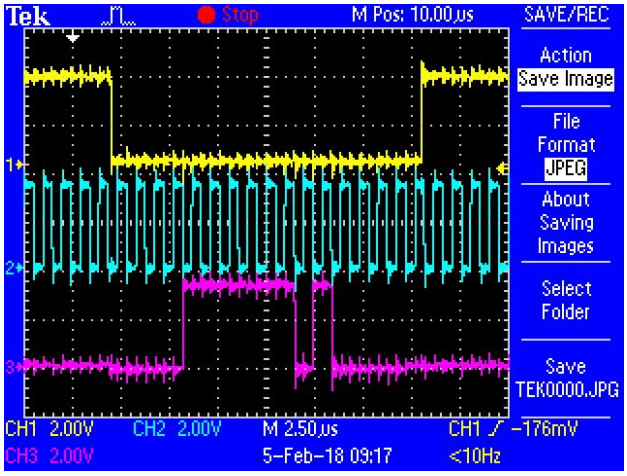
In this block diagram, inputs are clock signal and the feedback from PmodALS (light sensor plus ADC), which is after sending signals from outputs: chip select and signal clock. Finally, the eight bits data are sent out as the output.




Initially, 25MHz is divided to be 1MHz clock using in-module counter similarly to one in the seven segment module. Unfortunately, we cannot achieve that goal directly because the number to divide is not an integer, 12.5. The seven segment display ports are tied to the seven segment module earlier created in the preliminary

part so that the obtained result can be shown on the seven segment. From the light sensor manual documentation, after the chip select is enabled low, SDO, serial data out (master in slave out), sends three leading zeros, then eight data bits, and finally four following zeros. All the bits can be read at the rising edge of SCLK input. Furthermore, the eight data bits are ordered from most significant bit to least significant bit. After the data are put into buses, they are used in the instantiated module as bits to show on the seven segment. The data is read by left shift register. Because rising edges of 25MHz clock are relied for the always block, an another wire is used as a reference to keep track if the bit of SDO is already read or not. If the 25MHz clock is triggered when the SCLK is low, the bit is change to one to indicate that SDO is ready to be read for next rising edge of SCLK, which cannot be directly tracked. Otherwise, next condition, chip select enabled low, will be checked. A new variable is also introduced to keep track of the number of bit being read. Hence, if the chip select is enabled low, the module checks if the ready bit is set to 1 or not; if so, it will read SDO if the number of read bit is between 3 and 10 inclusively (the value starts from 0), increment the value of the number of bit being read, and change the ready bit to 0 to indicate that the bit has been read. The refresh rate for output on the seven segment display is 5Hz by using a counter and clock enable signal realizing 25MHz clock. Instead of update value directly every 200 ms, individual buses will keep an updated data from the peripheral by let chip select be a clock signal of at most one thirty-second of SCLK so that SDO can be fully retrieved. Then, once in 200 ms, these individual buses will be transferred to the variable which connects to seven segment display.

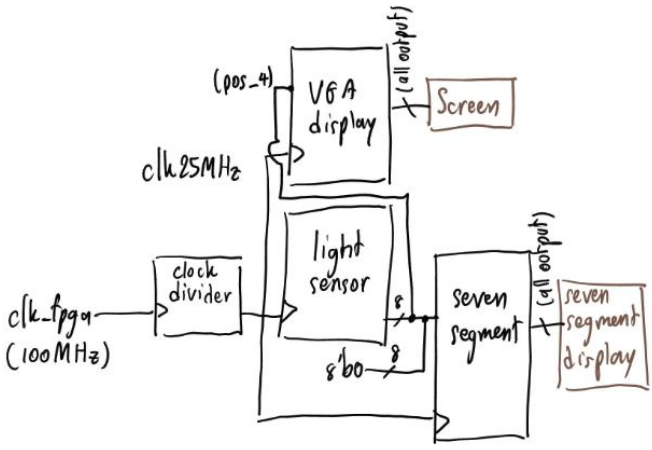
We can get the signals from the board shown in the picture beside. Yellow, cyan, magenta colors represent chip select, SCLK, and returned value. In this result, light sensor takes the flash from smartphone directly and closely; the value reported on the seven segment display is 0xFD. In the picture which also show that the signal is not as expected (the bit starts at fifth position), the light sensor value can be interpreted as 11111101 or 0xFD as shown on seven segment display.




The first encountered problem is the correctness of the module cannot be checked easily. This problem was solved by using oscilloscope to measure signals of chip select (CS), clock signal (SCLK), and master-in-slave-out (SDO). However, the difficulty occurs because the first used oscilloscope is broken and cannot be validated that easily. Next, using the same clock signal for all ‘always’ blocks is a confusing concept in my opinion even though we can reform my code to do so, but cause-unknown issue occurs; according to the documentation of light sensor, there are three leading zeros, but from trials and errors, the values of light sensor start at fifth position instead of forth position. The reason might be the falling edge of chip select signal and the rising edge of SCLK is not as expected, or the blocking and non-blocking assignments evaluate differently than expected. Last, the counter variable of clock divider is doubled or halved of the value supposed to be. Personally, we mostly fixed by trials and errors

so that the result is the same as planned. Nonetheless, all codes written are not only workable but also understandable.

Part 3: Combination




In this part, all three parts are combined together. However, the separate modules originally instantiate MMCM to change from 100MHz FPGA clock to 25MHz. The modules' inputs are changed to represent as 25MHz clock instead of 100MHz one, and the MMCMs in the modules are deleted. Furthermore, in the light sensor module, the embedded seven segment display is also removed; it will return 8-bit returned data instead.

In this part, we did not encounter any difficulty.

## Conclusion

We can accomplish in designing and implementing circuits to light sensor and VGA. Specifically, we learn how to communicate and connect with the peripheral on FPGA board. All the screen patterns can be achieved; serial-data-out data can be read and interpreted correctly. Although facing plenty of problems, we can gradually solving them point by point such as unexpected short clock due to miscalculation of period and frequency. In conclusion, we can successfully employ based modules from developers and understand how given peripherals work.

# Appendices