RGB LED Control

List of Authors (Jermaine Juarez, Tajwar Eram, David Sheridan)

Electrical and Computer Engineering Department

School of Engineering and Computer Science

Oakland University, Rochester, MI

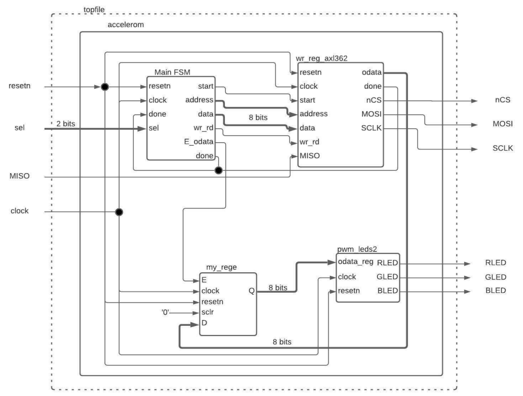
e-mails: jmjuarez@oakland.edu, tzeram@oakland.edu, dsheridan2@oakland.edu

Abstract—The purpose of this project is to create an RGB LED controller using a Nexys A7 board. The user will control the intensity of each color with the accelerometer (ADXL362) data on the board. The change in color is caused by the change in orientation from the change in the 8-bit data from the accelerometer.

Introduction

The Nexys A7 board has many things that can be utilized in the board to control its tri-color RGB LEDs. The accelerometer on the board could be used to control the RGB LEDs in the real world as they are important for communication protocols. Especially to warn individuals if the environment or orientation is hostile for the specific hardware it is being communicated with. The scope of this project will not go that far, however, a good start is to be able to move the accelerometer and allow a vast array of colors.

Utilizing knowledge from the curriculum, VHDL code to reference, and through further research, fundamental approaches to low- level-programming through Vivado will be used. This includes synthesis of the code, functional simulations, generating bitstreams, and finally programming the Artix-7 board. With the approach to coding, breaking the whole project into a block diagram is necessary which will be seen next in the report.

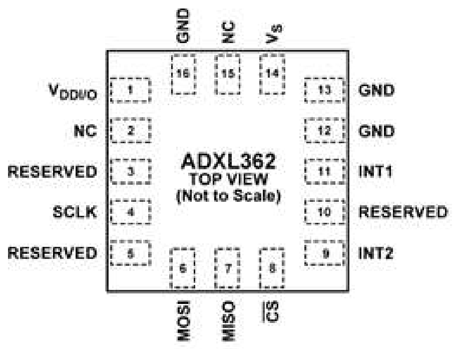


Specifically, as you see from the block diagram, the accelerometer, PWM, and FSMs are all components in the system. Each color will be dedicated to certain bits of accelerometer like the least significant bits related to the Blue LED and the most significant bits to the Red LED. This is the fundamental methodology of this RGB controller. The individual components and VHDL implementation will be gone into more in-depth in the next section.

Methodology

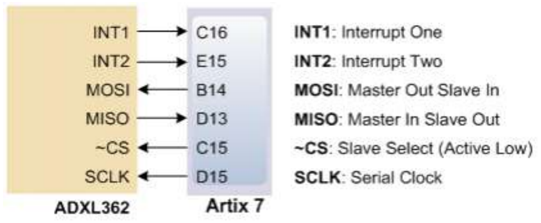
ADXL362 Accelerometer

The main objective of this project is to use certain hardware on the Nexys Artix-7 board to control the color and intensity of the tricolor LEDs though changing the orientation of it. Specifically, using the ADXL362 accelerometer. The pins of this hardware is seen below:



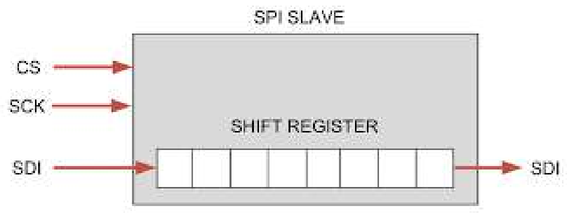
SPI

From these pins, you can see the various SPI pins needed for communication with the ADXL362 and the Nexys Artix 7. The pins MISO, MOSI, SCLK, and nCS are the SPI signals. The Artix 7 acts as a "Master" bus for the data and ADXL362 is the "Slave" bus that allows the data to go through. The two pictures below highlight what these pins exactly are.



INT1: Interrupt One
INT2: Interrupt Two
MOSI: Master Out Slave In
MISO: Master In Slave Out
~CS: Slave Select (Active Low)
SCLK: Serial Clock

[1], [3]



Shift Register Used

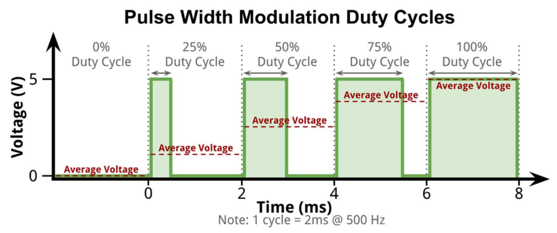
The second picture highlights the process of using a shift register to transfer the data to the Master bus. The specific file, which will be gone into more detail later in this report is called the "wr_reg_ADXL362" file. The reading of data from the ADXL362 then allowed the group to approach its color control.

Approaches to Color Control

The purpose of the orientation based color control, with the accelerometer, is to provide information about the x, y, and z axes. If a specific color is assigned to an axis, it gives a great variety of colors based on moving the board around. The intensity of each LED is altered based on the VHDL code written and referenced. Additionally, the selector switches are used to read a specific axis.

Pulse Width Modulation (PWM)

Pulse Width Modulation is used to control the brightness of the LEDs. This mimics what analog signals can do in a digital environment. Different duty cycles provide varying degrees of intensity which is useful for this project. These are consistent pulses that vary in width to create these various intensities [5]. This can be seen from the figure below:



It can be seen that the average voltage goes up proportionally with the duty cycle. Given lower duty cycles, the brightness is low. This is because the led flickers less frequently at lower duty cycles, giving the illusion of the brightness decreasing. Implementing this in VHDL would be possible with code that takes advantage of the clock on the board and setting specific frequencies to give quality variations. Making sure to map the correct variables to the .xdc files for the specific board is also important as the RGB LED must get the correct output from the code. This requires combinational circuits to be used with the addition of Finite state machines.

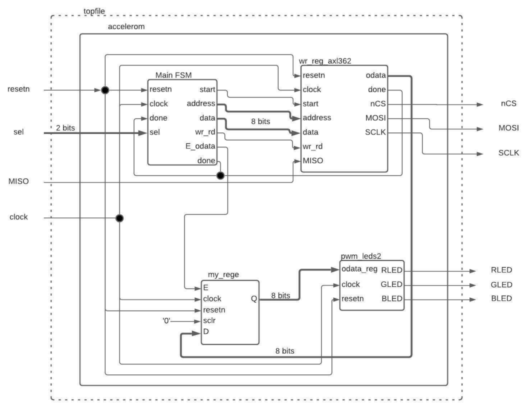
Finite State Machines (FSMs)

The FSMs are imperative to keep the project from having logical inconsistencies as different states are required for the different modes. For example, there should be some way for the counter to function based on how many counts are needed. This project specifically needed a modulo-8 counter which will be gone more in depth in the experimental setup section. FSMs also have outputs that cycle through until the full result is found. This is especially seen in the Accelerometer and PWM block. The ADXL362 sensor needs certain addresses to wake up the board and prepare for the measurement mode which will be explored further in the next section of this report. Additionally this will also put all these components together to make a functional RGB controller.

Experimental Setup

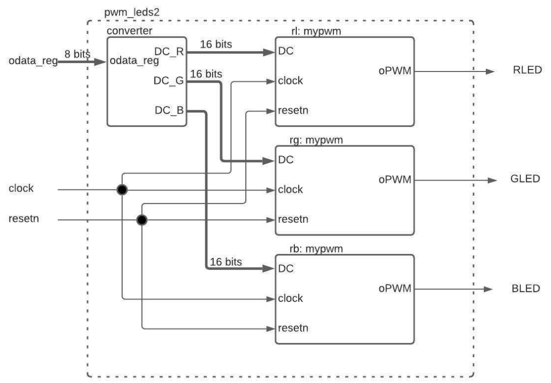
Implementing the components in VHDL and then testing it was essential to bring this RGB LED controller to life. Some of the most important parts of the testing in this project was to do with synthesizing the code for syntax errors. This then left the ".xdc" to be checked for any errors through generating the bitstream. When programming the board, it was able to be told if the system was working properly. The overall system is shown again below:

Overall System

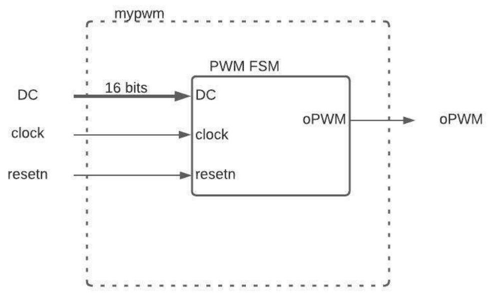


From this overall system, it can be seen that the top file, with all of the components, has 4 inputs and 6 outputs. One of the inputs is for the MISO (Master In Slave Out) which is an SPI signal. The "resetn" and "clock" were inputs for the top file and also other internal components. And finally, a two-bit selector for the 3 modes for the registers for the axes is an input. Half of the outputs are dedicated to the SPI signals nCS, MOSI, and SCLK while the other half is dedicated to the RLED, GLED, and BLED coming straight out of the internal PWM system. The block diagram can be seen in the next sub-section.

PWM LED system

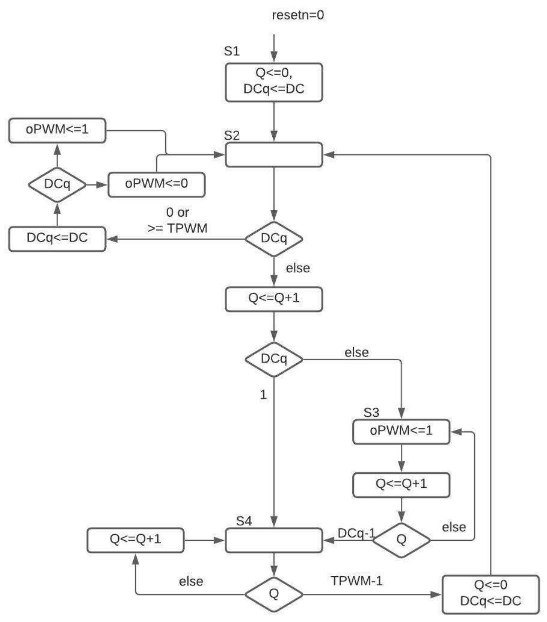


Along with the resetn and clock inputs, the PWM LED system requires the 8 bit data bus that comes from the register. The 4 most significant bits are used to calculate the duty cycle for the red LED, and the 4 least significant bits are used to calculate the duty cycle for the blue LED. The green LED uses a set 4 bits of "0001" to calculate the duty cycle for the green LED. The 4 bit values are then multiplied by the time period variable (TPWM). The TPWM variable is set to 50000. The product of the two variables is a twenty bit number, and the 16 most significant bits are sent to the mypwm block for the respective colors.



As previously mentioned, there is a mypwm block for each color of the LED. The single bit oPWM output is what is sent to the RGB LED, and the only component inside the block is the PWM FSM.

PWM FSM (pwm_led2.vhd)

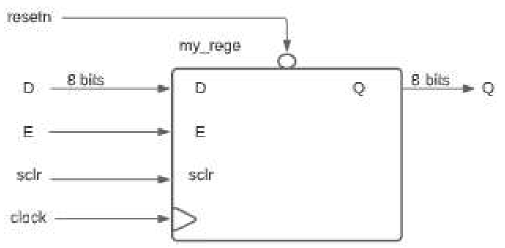


The PWM FSM is what controls the pulse of the LED. The first State resets the finite state machine. State two checks that the duty cycle is within the proper bounds. If the duty cycle is zero or above the TPWM value, the FSM stays in State two. While the duty cycle is zero, the oPWM value remains low. Otherwise, the oPWM output is high. If the duty cycle is within bounds, Q is incremented. If the duty cycle is one, the FSM goes directly to State four. Otherwise, it goes to State three. While in State three, the oPWM is high and Q is incremented. Q is then checked against the duty cycle value. If it is equal to one less than the duty cycle, the FSM goes to State four. If it is less than the duty cycle minus one, it remains in State three. State four increments Q until it reaches the TPWM value minus one, and sets the oPWM value stays at zero. Once that is reached, the Q value is reset, a new duty cycle is set, and the FSM goes back to State two. So the duty cycle value can be seen as a percentage, and while in State three the FSM counts out the "on" pulse of the PWM, while State four counts out the "off" pulse.

Accelerometer

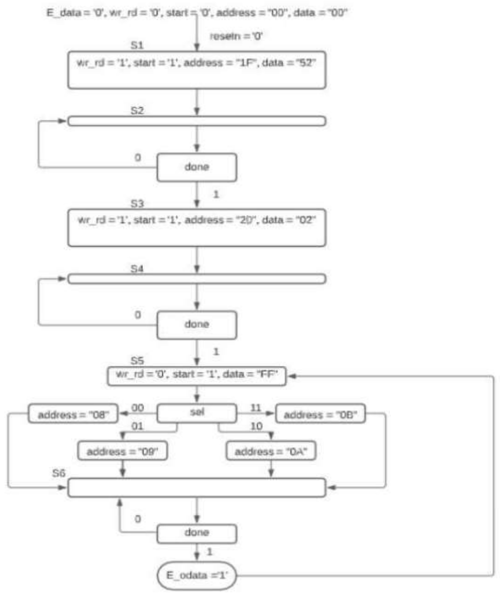
The accelerometer used is an ADXL362 accelerometer found on the board of the Nexys A7. The accelerometer measures data from the x-axis, y-axis, z-axis and produces the analog data into a digital 12-bit output. However, the 12-bit output is not really needed as the accuracy is not typically needed for many applications. Using 8-bit data is enough to send to the FSM to choose and measure the data of the different axis and to process it for the LED to change color [4]. The accelerometer diagram is very similar to the overall block diagram shown previously and consists of different components that will be talked about in the next sections.

8-Bit Register (my_rege.vhd)



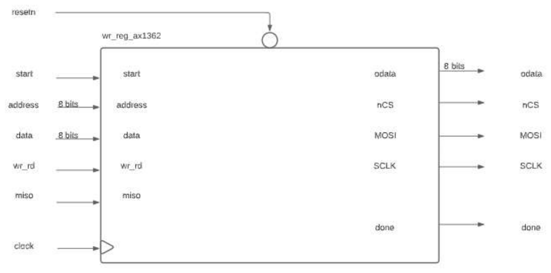
A register is created by multiple D-type flip flops and each flip-flop stores one bit. In the Accelerometer file, the register is what saves the values from the ADXL362. The enable input means that the register only gets activated when enable is high and when the clock input is high. Many of these registers can also make it possible to create a shift register which will be explored in the write/read register (wr_rd_reg_adxl362) for the ADXL362.

Accelerometer FSM

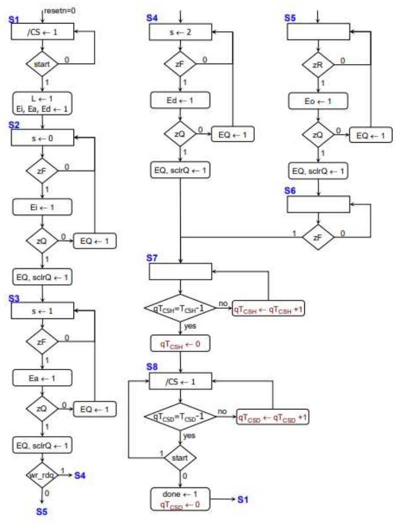

E_data = '0', wr_rd = '0', start = '0', address = "00", data = "00"
resetn = '0'
S1
wr_rd = '1', start = '1', address = "1F", data = "52"
S2
0
done
1
S3
wr_rd = '1', start = '1', address = "20", data = "02"
S4
0
done
1
S5
wr_rd = '0', start = '1', data = "FF"
address = "08"
00
sel
11
address = "0B"
01
10
address = "09"
address = "0A"
S6
0
done
1
E_odata ='1'

The Accelerometer Finite State Machine is what took the 8-bit data from the write/read register and processed it. Before the FSM is used many of the outputs are set to ‘0’ and “00” as an initial condition. When resetn = 0 the FSM begins and moves into State 1. In State 1, start = ‘1’, wr_rd = ‘1’, address = “0x1F”, and data = “0x52.” The FSM then moves into State 2 and repeats until done = ‘1.’ The FSM then moves into State 3 where it prepares for measuring. In State 3, start = ‘1’, wr_rd = ‘1’, address = “0x2D” which is called the POWER_CTL mode for measuring and data = “0x02” [4]. The FSM now moves into State 4 and repeats again until done = ‘1.’ These first four states prepare the FSM to measure and at State 5 it begins to take note of the data. At state 5, start = ‘1’, wr_rd = ‘0’, and data = “0xFF.” It then moves into the selector which chooses the address of data to access and what register to use. When address = “0x08’ it goes into the X-axis data register, when address = “0x09” it goes into the Y-axis data register, when address= “0x0A it goes into the Z-axis data register and when address = “0x0B” it goes into the status register [4]. After the selector chooses which register to access it moves into State 6 which repeats in State 6 until done = ‘1.’ When done = ‘1’, then E_odata = ‘1’, and the process repeats back up to State 5 and cycles back and forth between State 5 and State 6 until resetn = ‘0’ again. This would not be possible without the read/write register, which will be discussed below.

Read/write register (wr_rd_reg_adxl362)



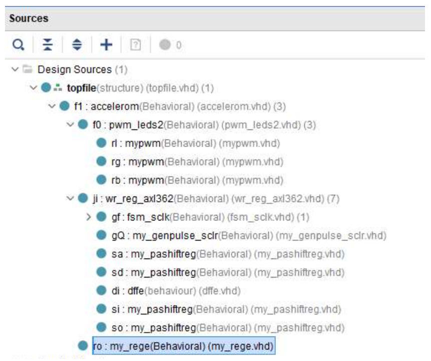
This register is a culmination of shift registers, a modulo-8 counter, normal registers, and an FSM. This FSM is complex and has a lot of outputs which just used the sample code because it fulfilled our purposes. The shift registers get their variables from the FSM. The purpose of this register is to provide the address, data, and read/write the 8 bits of the data [2]. The FSM from a higher level computer hardware design course was heavily referenced. The FSM can be seen below which is what was implemented into VHDL in Vivado through additional sample code.



Page 6 [2]

It would be very inefficient to create an FSM based on this read/write register, so it is very convenient to use this.

Now understanding the methodology, this can be seen from the sources bar in vivado which is underneath:



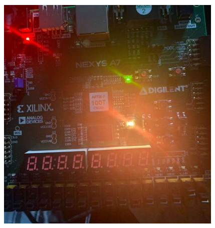
```
Sources
Design Sources (1)
  topfile(structure) (topfile.vhd) (1)
    f1 : accelerom(Behavioral) (accelerom.vhd) (3)
      f0 : pwm_leds2(Behavioral) (pwm_leds2.vhd) (3)
        rl : mypwm(Behavioral) (mypwm.vhd)
        rg : mypwm(Behavioral) (mypwm.vhd)
        rb : mypwm(Behavioral) (mypwm.vhd)
      ji : wr_reg_axl362(Behavioral) (wr_reg_axl362.vhd) (7)
        gf : fsm_sclk(Behavioral) (fsm_sclk.vhd) (1)
        gQ : my_genpulse_sclr(Behavioral) (my_genpulse_sclr.vhd)
        sa : my_pashiftreg(Behavioral) (my_pashiftreg.vhd)
        sd : my_pashiftreg(Behavioral) (my_pashiftreg.vhd)
        di : dffe(behaviour) (dffe.vhd)
        si : my_pashiftreg(Behavioral) (my_pashiftreg.vhd)
        so : my_pashiftreg(Behavioral) (my_pashiftreg.vhd)
      ro : my_rege(Behavioral) (my_rege.vhd)
```

It can be seen that the components that were discussed in this section are all seen and especially all the bottom files for the read/write register.

Results

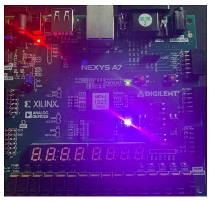
When programming the board, the color starts off as green, and then it changes based on the first two selector switches. The first one is for the Z-axis seen below:

Z-axis (Sel= “10”)

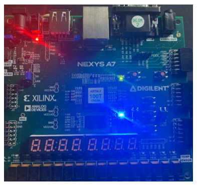


The variation of color can be seen from the selectors flipped to different things. Switch 0 is zero and switch 1 is one for the selector and this gives readings from the Z-axis register. There are other registers for other axes seen below with nice variation.

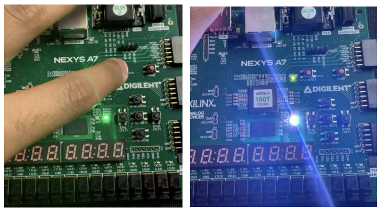
Y-axis (Sel= “01”)



X-axis (Sel= “00”)



It is important to note that there can be a soft register reset once the reset button is clicked as mentioned before [4]. This makes sure that the accelerometer FSM can go back to the first state and then instantly starts to read the values which can be seen below:



Conclusions

In conclusion, the accelerometer on the Nexys A7 board can be used to control the RGB LED. A great deal was learned in the creation of this project. A portion of the code used was obtained from example code used in class. Some was used for the PWM controller and some was used for the accelerometer control. The PWM used to control the LED is a useful tool that can be used to control different peripherals, like motors or speakers. Similarly, SPI protocol can be used to communicate with peripherals. While many new topics were learned, there were also issues encountered that were not overcome. The current version of the program uses only one axis at a time to control two colors of the LED, while the green portion stays constant. In future versions of the project, each axis of the accelerometer may be used to control an individual color of the LED. Also, the selector may be used to control the source of the input. Instead of just using the accelerometer, the program could use an accelerometer, a series of switches, or a temperature sensor to control the LED. All in all, the RGB LED could be improved for practical purposes, but exploring this provided amazing insight and can also indulge in even more control and variation in color.

References

1. Reference.digilentinc.com. 2020. [online] Available at: <https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-a7/nexys-a7_rm.pdf> [Accessed 4 November 2020].
2. Llamocca, D., 2020. *Unit 3 - External Peripherals: Interfacing*. [online] Available at: <http://www.secs.oakland.edu/~llamocca/Winter2020_ece4710.html> [Accessed 4 November 2020].
3. A. Brown, “Nexys A7 Reference Manual,” *Nexys A7 Reference Manual [Digilent Documentation]*. [Online]. Available: https://reference.digilentinc.com/reference/programmable-logic/nexys-a7/reference-manual. [Accessed: 25-Nov-2020].
4. Analog Devices, “ADXL362 Data Sheet.” Analog Devices, Norwood, 2019.
5. [1]"What is Pulse Width Modulation (PWM)? Definition, Basics, Generation and Detection Circuit and applications of Pulse Width Modulation - Electronics Coach",[Online].
6. *Pulse Width Modulation with analogWrite*. [Online]. Available: http://robotic-controls.com/book/export/html/57/. [Accessed: 12-Dec-2020].