Tim Neiser

12-10-15

ECE 287

Final Project Report

My project was to operate Adafruit’s Neopixel Lights using Altera’s DE2 board. Before it is possible to start getting into the logic to make the lights work, the hardware must be understood. These lights require three connections to them: a 5v line, a ground line, and a serial addressing line. In order to have enough current to power these lights, I used an ATX power supply from an old computer. Since the DE2 board can only output 3.3v through its digital pins and the serial addressing line requires 5v, I needed to use the SN74LS32N chip which has multiple OR gates. I used the board’s ground and non-addressable 5v pin to power the OR gate and I used my addressable digital output pin as the input for the OR logic along with the ground, so that the output of the OR gate would be a 5v if the input was 3.3v, and it would be 0v if the input was 0v. The output of the OR gate was then connected to the serial addressing line to the board. Now that all of the hardware is done, it is time for the details on how data is transferred.

The serial addressing line for the lights must be used to transfer data from the microprocessor to each light. Each light reads the first 24 bits (8 for each red, green, and blue), and then sends every bit after the first 24 to the next light. This is done so that theoretically there can be as many lights in the strand as wanted. Each bit has its own timing so that the lights can tell the difference between one bits and zero bits. A one bit is sent by having the serial line high for 800 ns and then low for 600 ns. A zero bit is sent by having the serial line high for 360 ns and then low for 700 ns. Once the data is complete for one strand, the serial line must be off for 50,000 ns. For example, if you were just trying to operate one light, you would send 24 total bits and then pause for 50,000 ns. If you were trying to operate 240 lights, like in my project, you would send 5,760 bits in a row, and then pause for 50,000 ns.

So now that it is clear how data is transferred, it’s time to discuss the logic. The first bit that is read is the most significant bit of an 8 bit number that tells the light at what intensity to shine red. An 8 bit number representing 0 means have no red and an 8 bit number representing 255 means shine the maximum amount of red. The next 8 bits read are for green and then the next 8 bits are for blue. So in my code I first had logic to declare vectors that represented each 8 bit number for red, green, and blue. The logic basically says that if one switch is on, make a seasonal lighting scheme such that the lights alternate between red and green. This is done by using counters and timers such that if each individual light is an odd or even number in a specific time frame then the light will either have only the most significant bit of green be a one or only the most significant bit of red to be one. Then, if that switch is not on, if any of the three switches that represent red, green, or blue are on, then each light on the strand will represent the same color but at a different brightness. Any combination of the switches makes a different color, for example, if the red and blue switches are on, the color purple is shown. If red, green, and blue are on, the color white will show. It’s really hard to describe these light functions, so I encourage you to look at my video that I posted if these functions are unclear. If anyone were to take this project and modify it for these lights and these boards, the only chunk of code that should be changed is this chunk where the values of the red, blue, and green vectors are changed so that the user can make any light combination that they want.

The switch statement is to not be changed as it is necessary for the lights to work properly. Case A is when the serial addressing line is sending the high signal to the lights. There is an if statement that says if the signal being transmitted is a one, then send only send a high value until the timer equals the time that is needed to send a one. If it is not a one, then send the high signal until the timer is equal to the time needed to transmit a 0 is needed. Once the timer is equal to the time needed, then it switches to case B. Case B does the same as A, but instead of sending the serial high signal for each bit, it sends the serial low bit. B also says if all 8 bits of red are sent, then it switches to case D. Cases D and E are the same as A and B, except that it is for the color green. Once all of the green’s bits are sent, it goes to case F and G which is blues bits. Once all of blue’s bits are sent, it goes to case A if it hasn’t cycled through this process 240 times (the number of lights on the strand), but if it has cycled through those states 240 times, then it goes to case C which causes a 50,000 ns pause, thus completing the lighting of the whole strand once. After case C, it starts over at case A.

Basically, this project uses logic based on switches to determine what the values of each light should be and then I use timing by using a case statement and state variables to send 3 values represented by 8 bit numbers to each light representing red, green, and blue by using serial addressing. It was a little challenging starting from scratch and working on my own without any sample code, but after I gave it a lot of thought and time figuring it out I was able to succeed at making this work. There was only one website that I referenced during this whole project, and that website gave a breakdown on how the lights work and it gave the data sheet, but I didn’t use any code because it was in a different language. I will provide the links to this page and my video on the lights below.

Website Referenced:

<http://wp.josh.com/2014/05/13/ws2812-neopixels-are-not-so-finicky-once-you-get-to-know-them/>

Video of Lights Working:

<https://youtu.be/eH3SBPpNDuA>