

Chapter 8

Final Design Project II

8.1 Section Overview

It is very important to be able to actually use the concepts learned in this course to create real systems and designs. This lab presents a design problem that will use the information and selected aspects gathered from previous labs.

8.2 Objective

Design an infrared remote control for your TekBot. This will use serial communications to communicate between two different d_logic boards. One of the boards will transmit a signal from three buttons used to control the TekBot and another board will receive and decode the signal.

Serial communications is the process of sending data one bit at a time, sequentially, over a communications channel or computer bus. Often data is sent in sets that include a start bit sequence to let the receiving system know when to start looking for the data. These start bits can be one bit or many bits long. It is up to the designer to decide what is appropriate for the application. Start bits can be any sequence of bits that the receiver is designed to look for. An example of a set of data sent by the transmitter might look like this:

010XXXXXX

Here, "010" are the start bits and the X's represent the data being sent. The number of bits sent out is always the same and the receiver will always be looking for the sequence of start bits. Once the set of bits being sent to the receiver matches the defined start sequence, the receiver will begin taking the next set of bits sent as data to be used. A possible check to make sure the data is valid would be to send the three bits from the switches along with those bits inverted. The check would be to add the two sets of three bits and see if the answer is 111. If it is, you know the data is valid.

8.3 Materials

1. Xilinx ISE 12 software (Currently installed on the lab computers)
2. Two Digital Logic Boards (d_logic.2 board)
3. Working Section 3 logic
4. A working TekBot with the motor control board
5. The ECE 271 textbook, Digital Design and Computer Architecture by David and Sarah Harris

8.4 Procedure

There are 5 steps to digital logic design:

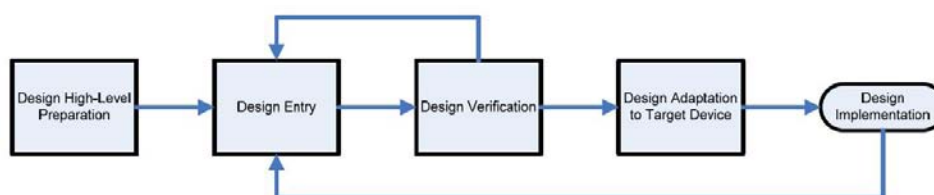


Figure 8.1: Use this process for designing the final project.

1. *High Level Preparation:* Section 8 requires the design and implementation of two different sections of digital logic. One section will be the transmitter logic and the other is the receiver logic. The transmitter logic will be programmed onto one of the d_logic boards with three buttons being used as inputs and the infrared LED as the output. The signal from the buttons will need to be encoded into long and short pulses shown in Figure 8.2. For an application like this, the short pulse will represent a 0 and the long pulse will represent a 1. This will enable the receiver to decode the signal into 1's and 0's. The receiver will consist of the infrared receiver as an input and three outputs to control the TekBot motors using Section 3.

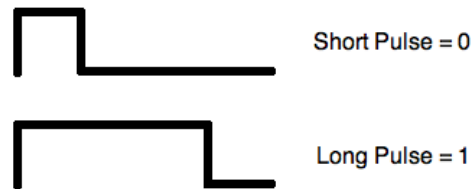


Figure 8.2: Example of long and short pulses.

Transmitter: The transmitter will consist of three blocks: a counter, a shift register, and a state machine. The block diagram is shown in Figure 8.3.

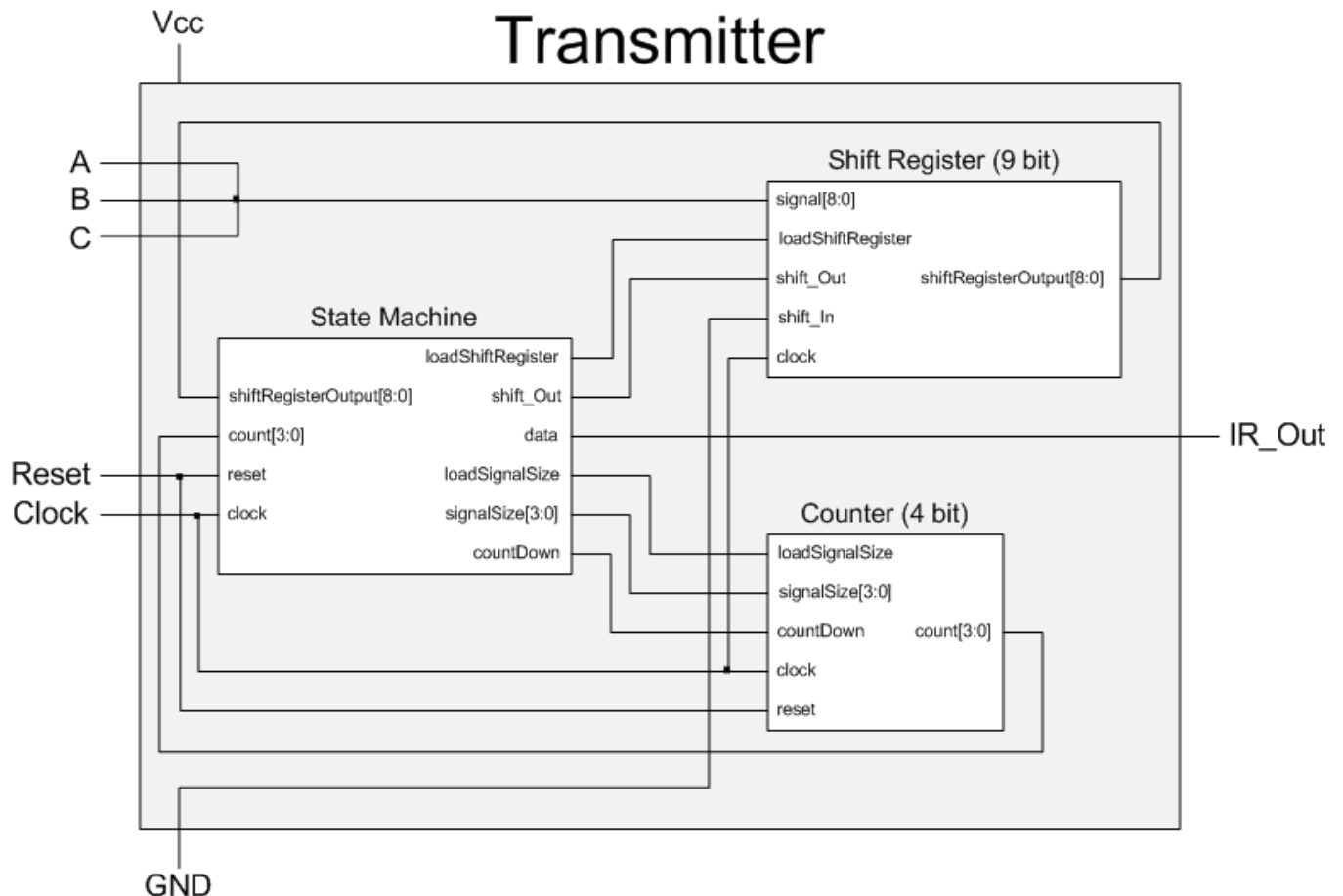


Figure 8.3: Transmitter Block Diagram

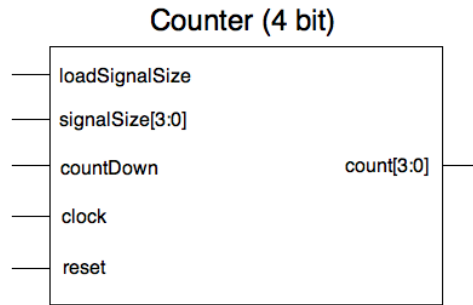


Figure 8.4: Counter Block

The counter, Figure 8.4, just counts to see when the shift register is empty so it can reset. It can either start at the number of bits the shift register can hold and count down, or start at 0 and count up.



Figure 8.5: Shift Register Block

The shift register, Figure 8.5, will store the values to be sent to the state machine. It will send the signal to the state machine one bit at a time until it has sent the entire signal. Then it will reset to all 0's and start over.

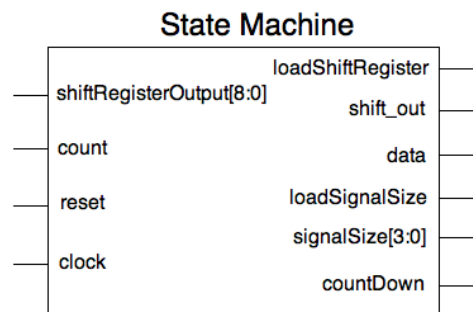


Figure 8.6: State Machine Block

The state machine, Figure 8.6 with a state diagram in Figure 8.7 will take the signal from the buttons and encode it into a series of long and short pulses based on the output bit of the shift register. These pulses will be sent to the receiver via infrared. The transmitter will include several variables used throughout the state machine to switch states and reset the counter and shift register. CountDown is used to tell the counter to decrease the count by 1. LoadSignalSize tells the counter to set the count to the signal size. LoadShiftRegister tells the shift register to load the signal. Shift.out tells the shift register to shift a bit out and shift in a 0.

The shift register will need to hold nine bits. The first three bits are the start bits, the middle three bits are the button inputs, and the last three bits are the inverted button inputs to be used as a checksum.

States 1 through 6 in the state machine are used to create pulses to send to the receiver. States 1, 2, and 3 will always send a 1. States 4, 5, and 6 will always send a 0. This will make the output be 1110 for a 1 and 1000 for a 0.

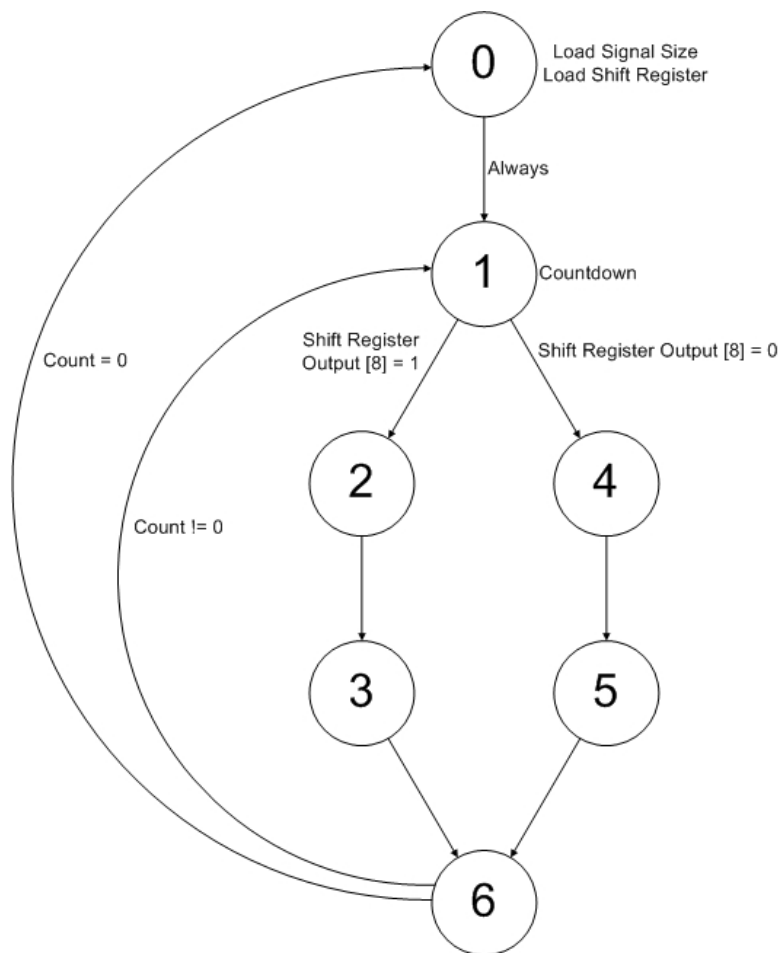


Figure 8.7: Transmitter State Machine

Receiver: The receiver will consist of 5 blocks: a decoder, an error check, a shift register, a counter, and a state machine. The block diagram is shown in Figure 8.8.

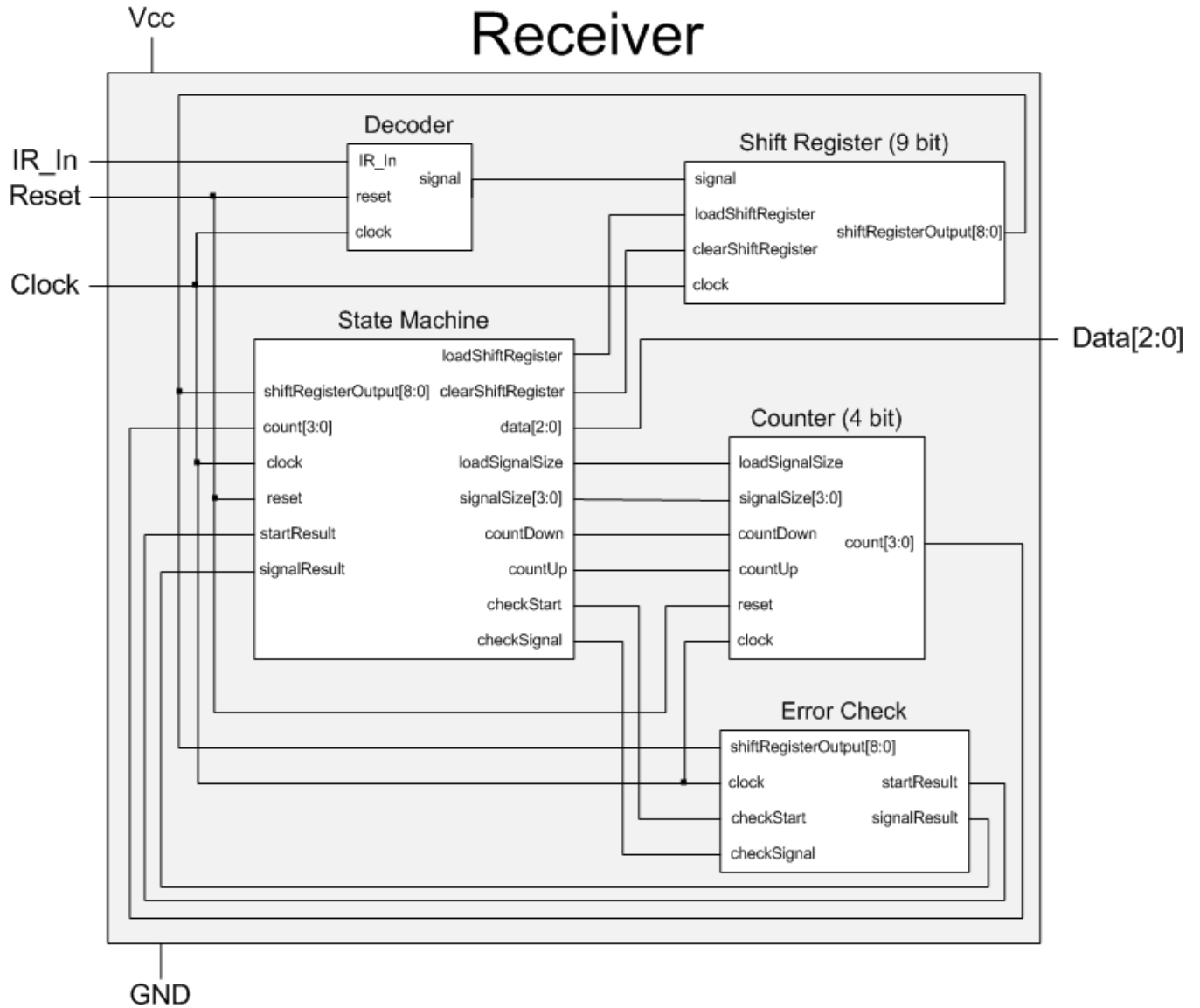


Figure 8.8: Receiver Block Diagram

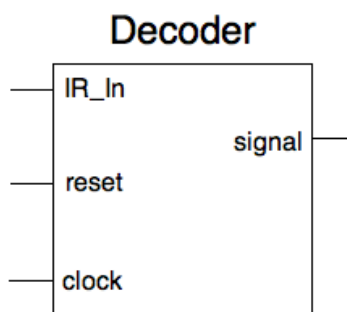


Figure 8.9: Decoder Block

The decoder, Figure 8.9, will decode the serial signal of pulses to 1's and 0's. In this case the input signal is IR.In and is sent into a shift register.

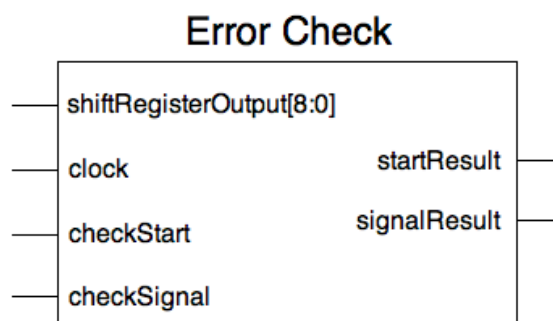


Figure 8.10: Error Check Block

The error check, Figure 8.10, will be used by the state machine to compare the first three bits of the signal (the start bits) until they match the pre-determined sequence. After the entire signal has come through, it will check to make sure that the second three bits of the signal and the third three bits of the signal sum to 111.

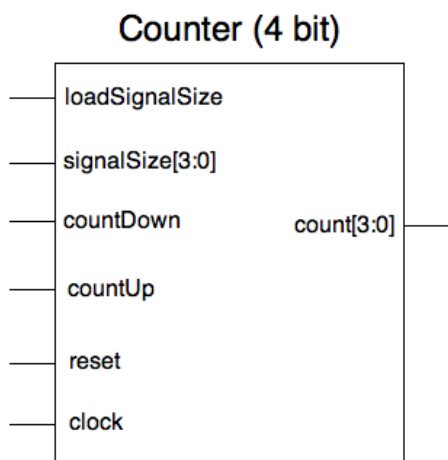


Figure 8.11: Counter Block

The counter, Figure 8.11, will count the number of bits in the shift register in the same manner as the counter for the transmitter.



Figure 8.12: Shift Register

The shift register, Figure 8.12, will hold the signal sent by the transmitter and shift it out to the state machine. The clearShiftRegister input is a signal sent from the state machine that will clear all the data from the shift register so it can start fresh.

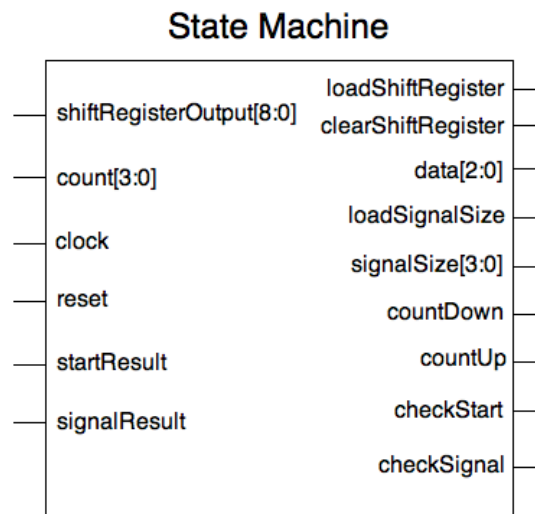


Figure 8.13: State Machine

The state machine, Figure 8.13, with a state diagram in Figure 8.14 will take the signal from the shift register (shiftRegisterOutput) and check to make sure the start bits are correct before the shift register shifts in more data. The state machine will know to check the start bits because the count will be 3 and will tell the error check to start by setting checkStart high. It will need to check when the shift register is holding three bits and continue checking until the start bits are correct. After the start bits are correct (startResult is high), the state machine will have the shift register collect the rest of the data until the entire signal has been received. The state machine will check to see if the two separate sets of data bits can be added together to make 111 and then output the signal to the Section 3 logic. Most of the inputs and outputs have the same function as their equivalents in the transmitter state machine.

The Receiver will output a bus (Data 2:0) of three bits that will be equivalent to which buttons are being pressed. They can then be sent to the remote control logic from Section 3 to control the TekBot.

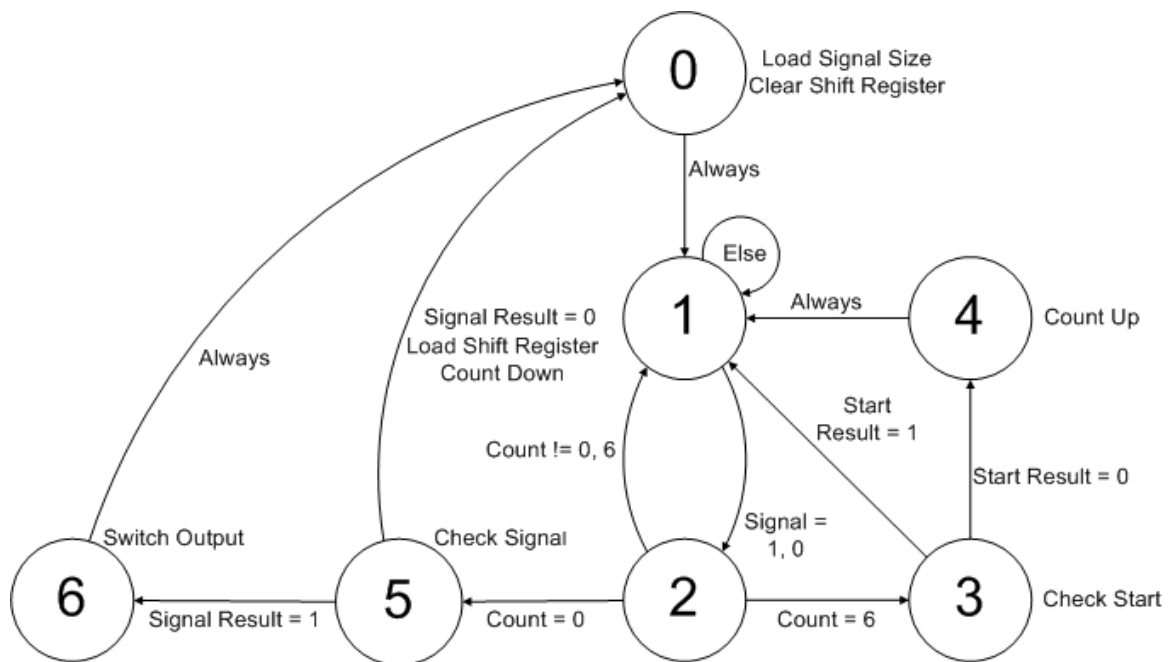


Figure 8.14: Receiver State Machine

2. *Design Entry:* Implement the preceding block diagrams. Remember that the transmitter and receiver need to be on separate d_logic boards.
3. *Design Verification:* Use the instructions in Section 3 to simulate the design to verify correct operation of the digital logic.
4. *Design Adaptation to Target Device:* Follow the same process for using PACE and iMPACT as in Section 2.
5. *Design Implementation:* Program the Xilinx CPLD's with the Universal Programmer software.



TA Signature: _____
 (TekBot functions properly & work is displayed)

8.5 Study Questions

1. Include a detailed block diagram of Section 8, the RTL Schematic with the blocks compressed, the RTL Schematic with the blocks expanded, and a copy of the Verilog source.
2. What was the toughest aspect of ECE 272? What should be changed or added to the ECE 272 manual to make this course better?
3. What would you like to explore further about Xilinx or Digital Logic Design?
4. What section of ECE 272 did you dislike the most? Why?
5. What was your favorite section of ECE 272? Why?

8.6 Challenge - Extra Credit

Enjoy your break!