ECE 241 **Lab 09B Report** **Spring 2021**

Name: Keegan Bean **Lab Section 14B**

Objective: Gain experiences programming, emphasizing the use of code developed thus far in the class. This lab will involve building a binary clock, employing the LED display and board from ECE 241. Its basic functionality will be adding the Part 1 clock, the ability to set the clock, and the display of the clock on the LED display.

Prelab: Code was uploaded to the website.

Part 1) Completed in lab09A. A clocked was created and the program added the ability to check for short and long presses. This code was verified by the TA on the LCD.

Part 2) The code was altered to allow for the ability to set the clock as well as display the clock on the LED as well. This code was debugged and checked off by the TA

Appendix A: Description of clock functionality.

The clock begins running when the program is running, and the state is CLOCKRUNNING. The clock will increase whenever it remains in this state.

Each time there is a long press, or the button is pressed for more than 500 milliseconds the clockState will change in the order of CLOCKRUNNING, CLOCK_SET_HOURS, CLOCK_SET_MINUTES, then back to clock running.

Each time there is a short press, or the button is pressed for less than 500 milliseconds the Hours, Minutes, or Seconds will be updated by one depending on the state. Nothing happens in CLOCK_RUNNING, then the rest update the above states in the same order. This is how the clock is set.

The clock is sent to the Lcd and Led using SendClock(). The LED displays the clock in the form of a binary clock. Columns 8 and 7 are hours, 5 and 4 are minutes, and 2 and 1 are seconds. The binary moves up from bottom to top and the time can be calculated from left to right using binary conversion of each of the according columns.

Appendix B: How I tested the clock.

First: I ran the program and made sure that the clock began running at 1. Then I proceeded to do a long press sending it to set hours and made sure that the clock did not run, then doing another long press sending it to set minutes and checking if the clock did not run, then another long press sending it to set seconds and checking if the clock did not run, finally I did another long press and checked that it began running on both the LED and LCD again.

Second: I checked that the clock updated by increments of one on the LCD and LED when it is any of the set states and there is a singular short press. I also checked that short presses do nothing when the clock is clock running.

Third: I would start the clock, wait, and do a long press to check that the LED and LCD were correct.

Fourth: I checked the wraparound points on the LED and LCD for the clock setting it at times that are close to becoming zero again making sure the transition happened properly on both.

Appendix C: Complete documentation of code

Lab09.ino:

```
#include "ClockBasics.h" //Header file for ClockBasics
#include "ButtonDebounce.h" //Header file for ButtonDebounce
#include <SPI.h> //Include the SPI library for the LED grid

unsigned long ClkTimer; //Timer for the clock

void setup() { //Setup for the main program
  ButtonInitialize(); //Call the initialize for buttonDebounce
  LcdDriver.begin(16, 2); //Initialize the display
  LcdDriver.clear(); //Clear the display
  Serial.begin(9600); // Initialize the serial to 9600
  pinMode(A0, OUTPUT); //Set the CS as an output
  digitalWrite(A0, HIGH); //Set CS high
  SPI.begin(); //Start the SPI
  SPI.beginTransaction(SPISettings(8000000, MSBFIRST, SPI_MODE0)); //Configure SPI hardware
  SW_SPI_16(MAX7219_TEST + 0x01); //Turn on all the LEDS
  delay(100); //One time we can use a delay
  SW_SPI_16(MAX7219_TEST + 0x00); //All LEDS off
  SW_SPI_16(MAX7219_DECODE_MODE + 0x00); //Disable BCD mode
  SW_SPI_16(MAX7219_BRIGHTNESS + 0x03); //Use lower intensity
  SW_SPI_16(MAX7219_SCAN_LIMIT + 0x0f); //Scan all digits
  SW_SPI_16(MAX7219_SHUTDOWN + 0x01); //Turn on chip
 } //End setup

 void loop() { //Loop for main program
  if (millis() - ClkTimer >= 1000) { //If 1 second has passed
   if (clockState == CLOCK_RUNNING) //If the clock is running
    UpdateClock(); //Then update the clock
   SendClock(); //Send the information for the clock
   ClkTimer += 1000; //Update the ClkTimer
  }
  switch (ButtonNextState(digitalRead(4))) { //Switch reading the button from pin 4
   case 2: //If the press is SHORT
    IncreaseClock(); //Increase the clock with a short press
    SendClock(); // Send the clock information
    break;
   case 3: // If the press is LONG
    MoveClockState(); //Move to set the next state with a long press
    SendClock(); // Send the clock information
    break;
  } // end of switch
 }//End of the loop
```

```

```

ButtonDebounce.h:

```
#ifndef ButtonDebounce_H
#define ButtonDebounce_H

#include "Arduino.h"

unsigned long ButtonTime; //Timer for the button
//Set up button state
enum ButtonState {buttonIdle, buttonWait, buttonLow} state;
//Set the initial conditions for the button
void ButtonInitialize() { //Set inputs and outputs
  pinMode(4, INPUT);
  pinMode(13, OUTPUT);
  state = buttonIdle; //Set the button to idle
}

int ButtonNextState(int Input) { //The void for the change in button states
  switch(state) { //Switch based on button state
    case buttonIdle: //case for button Idle
      if (Input == LOW) {
        ButtonTime = millis(); //Begin ButtonTime
        state = buttonWait;
        digitalWrite(13, HIGH); //Set the output or pin 13 HIGH
      }
      break;
    case buttonWait: //Case for buttonWait
      if (Input == HIGH) { //If the input is high
        state = buttonIdle; //Set the state back to idle
        digitalWrite(13, LOW); //Set it back to low
      }
      else {
        if(millis() - ButtonTime >= 5) { //If 5 milliseconds have passed
          state = buttonLow; //Set the state to low
          digitalWrite(13, LOW);
          return 1;
        }
      }
      break;
    case buttonLow: //If low
      if(Input == HIGH) {
        state = buttonIdle;
        if(millis() - ButtonTime >= 500)
          return 3; //If the button is pressed for longer than 500 milliseconds
        return 2;  //If the button is pressed for less than 500 milliseconds
      }
      break;
  }
  return 0; //Return 0
}

#endif //End of the header file

```

ClockBasics.h:

```
#ifndef ClockBasics_H
#define ClockBasics_H

#include <SPI.h>
#include <LiquidCrystal.h> //Library for the LCD

LiquidCrystal LcdDriver(A5, A4, 5, 6, 7, 8); //Set the correct pins
//MAX7219 SPI LED Driver
#define MAX7219_TEST          0x0f00 //Display in Test mode
#define MAX7219_BRIGHTNESS    0x0a00 //Set brightness of display
#define MAX7219_SCAN_LIMIT    0x0b00 //Set scan limit
#define MAX7219_DECODE_MODE   0x0900 //Sets chip to accept bit patterns
#define MAX7219_SHUTDOWN      0x0C00 //Code for shutdown chip

// Variable used as clock settings.
int CLK_Hours, CLK_Minutes, CLK_Seconds;
//Variables for the display to LED grid
int UpperDigit, LowerDigit;


void SW_SPI_16(int data){ //Function to display to the LED display grid
  PORTC &= ~0x01; //Set CS low
  SPI.transfer16(data); //Call the transfer for data
  PORTC |= 0x01; //Now set CS to high
}


// This function is to be called every second
// to update the clock represented by the
// global variables Hours, Minutes, Seconds
void UpdateClock()
{
  // Check if Seconds not at wrap point.
  if (CLK_Seconds < 59)
  {
    CLK_Seconds++; // Move seconds ahead.
  }
  else
  {
        CLK_Seconds = 0; // Reset Seconds
       // and check Minutes for wrap.
       if (CLK_Minutes < 59)
       {
         CLK_Minutes++; // Move seconds ahead.
       }
       else
       {
        CLK_Minutes = 0; // Reset Minutes
        // check Hours for wrap
        if (CLK_Hours < 23)
        {
          CLK_Hours++;// Move Hours ahead.
        }
        else
        {
          CLK_Hours = 0;// Reset Hours
        }// End of CLK_Hours test.
       } // End of Minutes test
  } // End of Seconds test
} // end of UpdateClock

//enums for the state of the clock
enum ClockStates { CLOCK_RUNNING, CLOCK_SET_HOURS, CLOCK_SET_MINUTES, CLOCK_SET_SECONDS};
ClockStates clockState = CLOCK_RUNNING;

 //Send Hours, Minutes and Seconds to a display
 void SendClock()
 {
  LcdDriver.clear(); //Clear the display
  LcdDriver.setCursor(1,0); //Set the display to the appropriate location
  if (CLK_Hours < 10)
  {
    LcdDriver.print("0");
  }
  LcdDriver.print(CLK_Hours); // Then send hours
  LcdDriver.print(":"); // And separator
  // Check for leading zero on Minutes.
  if (CLK_Minutes < 10)
  {
    LcdDriver.print("0");
  }
  LcdDriver.print(CLK_Minutes); // Then send Minutes
  LcdDriver.print(":"); // And separator
  // Check for leading zero needed for Seconds.
  if (CLK_Seconds < 10)
  {
    LcdDriver.print("0");
  }
  LcdDriver.print(CLK_Seconds);

  switch (clockState) { //Switch for moving the cursor

    case CLOCK_RUNNING:
      break;
    case CLOCK_SET_HOURS: //For setting the hours
      LcdDriver.setCursor(0, 0); //Set the cursor next to hours
      break;
    case CLOCK_SET_MINUTES: //For setting minutes
      LcdDriver.setCursor(3, 0); //Set the cursor next to minutes
      break;
    case CLOCK_SET_SECONDS: //For setting Seconds
      LcdDriver.setCursor(6, 0); //Set the cursor next to seconds
      break;
  } // end of switch

  LcdDriver.cursor(); //Set the cursor
  LcdDriver.blink(); // Make the cursor blink
  //Displaying to LED grid
  SW_SPI_16(0x0600 + 0x0000); //Clear column 6
  SW_SPI_16(0x0300 + 0x0000); //clear column 3
  UpperDigit = CLK_Hours / 10; //UpperDigit and LowerDigit for CLK_Hours
  LowerDigit = CLK_Hours - 10*UpperDigit;
  SW_SPI_16( 0x0800 + (UpperDigit & 0x0f)); //UpperDigit to Column 8
  SW_SPI_16(0x0700 + (LowerDigit & 0x0f)); //LowerDigit to Column 7
  UpperDigit = CLK_Minutes / 10; //UpperDigit and LowerDigit for CLK_Minutes
  LowerDigit = CLK_Minutes - 10*UpperDigit;
  SW_SPI_16( 0x0500 + (UpperDigit & 0x0f)); //UpperDigit to column 5
  SW_SPI_16(0x0400 + (LowerDigit & 0x0f)); //LowerDigit to column 4
  UpperDigit = CLK_Seconds / 10; //UpperDigit and LowerDigit for CLK_Seconds
  LowerDigit = CLK_Seconds - 10*UpperDigit;
  SW_SPI_16( 0x0200 + (UpperDigit & 0x0f)); //UpperDigit to column 2
  SW_SPI_16(0x0100 + (LowerDigit & 0x0f)); //LowerDigit to column 1
}//End of SendClock()



//Function used to move through
//the states of setting the clock
void MoveClockState()
{
  switch (clockState)
  {
    case CLOCK_RUNNING: //From Running
      clockState = CLOCK_SET_HOURS; // go to set the hours
      break;
    case CLOCK_SET_HOURS: // From set hours
      clockState = CLOCK_SET_MINUTES; //go to set the minutes
      break;
    case CLOCK_SET_MINUTES:
      clockState = CLOCK_SET_SECONDS; //go to set the seconds
      break;
    case CLOCK_SET_SECONDS:
      clockState = CLOCK_RUNNING; //set back to running
      break;
  } // end of switch
} // end of MoveClockState()

//Function that increase clock
//based on the state of the clock.
```

```
void IncreaseClock()
{
  // interpret input based on state
  switch (clockState)
  {
    case CLOCK_RUNNING: // Nothing to change if running.
      break;
    case CLOCK_SET_HOURS: //In this state
      CLK_Hours++; // the Hours in be incremented
      if (CLK_Hours > 23) // Watch for wrap
        CLK_Hours = 0;
      break;
    case CLOCK_SET_MINUTES: // In this state
      CLK_Minutes++; // the Minutes in be inremented
      if (CLK_Minutes > 59) // Watch for wrap
        CLK_Minutes = 0;
      break;
    case CLOCK_SET_SECONDS: // In this state
      CLK_Seconds++; // the Seconds in be incremented
      if (CLK_Seconds > 59) // Watch for wrap
        CLK_Seconds = 0;
      break;
  } // end of switch
} // end of IncreaseClock()

#endif //End of ClockBasics header file

```