Custom MIDI Controller

VOLUME 1: Proof of Concept

Austin Stockwell
April 2020

Acknowledgments

Various resources were used to help create this MIDI controller. The end result was a device that was suited to my specific musical needs. I would like to thank all of those who helped create the final product! [1] [2] [3] [4] [5]

Contents

List of Figures

1 Introduction

This project started as a desire to create a custom MIDI controller for use in my music studio. I often use external hardware and software plugins that have parameters that can be controlled in real time via the MIDI protocol. I therefore decided to design a 'proof of concept' for a MIDI controller that had the ability to send CC (Control Change) MIDI data through a 5-pin cable to such devices.

What makes this project unique is the fact that it does not utilize any of Arduino's built-in MIDI libraries. For this reason, the project served as a great platform for learning the MIDI protocol and how it is used to establish serial communication between MIDI-enabled devices.

2 Features

The following features are integrated into the MIDI controller:

- 2 Storable Presets
- MIDI Control Number Editing
- MIDI Channel Editing
- LCD Display
- LED Indication Lights
- Push Buttons
- Potentiometers
- 5-Pin MIDI Cable Connectivity

3 Bill of Materials

The components necessary to build the device are listed below:

- (1) Arduino MEGA
- (1) 5-Pin MIDI Jack (Female)
- (1) 1602 Liquid Crystal Display (LCD)
- (7) LEDs
- (4) 10K OHM Potentiometers
- (4) Momentary Push Buttons (Normally-Open)
- (11) 10K OHM Resistors
- (1) 220 OHM Resistor
- Wire
- Breadboard

4 Hardware

4.1 Hardware Design Philosophy

The design philosophy of this MIDI controller was 'simplicity'. Therefore, many features found on other MIDI controllers (such as an abundance of presets, preset naming, MIDI Note functionality, MIDI Clock functionality, etc) were omitted.

The end result was a controller that was extremely fast and easy to use due to this simple hardware design. The user can swap between the two presets (and therefore send different MIDI Channel / Control Numbers with the same knob) with the touch of a single button. Editing is also extremely simple – the user can assign a new value to the desired control within seconds.

4.2 Peripherals

The user interacts with the device using the following peripherals:

- 1602 LCD
- Pushbuttons
- Potentiometers
- LEDs

4.2.1 1602 Liquid Crystal Display

An LCD display was incorporated for MIDI Channel / Control Number editing and display of the current preset. The included Arduino Library (LiquidCrystal.h) was used to interface with the display.

4.2.2 Pushbuttons

The following pushbuttons are incorporated into the design:

- Preset 1 Button
- Preset 2 Button
- Edit Button
- Commit Button

The Preset 1 and Preset 2 Buttons are used to switch between the active preset in SESSION MODE. The Edit Button is pushed when the user wishes to modify the controls of the currently selected preset. This button is pushed once to enter EDIT MODE. Once in EDIT MODE, the user pushes the Edit Button again to toggle between MIDI Channel / Control Number Editing. The Commit Button is used when the user wishes to save the new value of the control into the Arduino's internal EEPROM memory.

4.2.3 Potentiometers

The following MIDI data is transmitted serially while a knob continues to move:

- MIDI Channel (1-16)
- Control Number (0-127)
- Control Value (0-127)

Each potentiometer has its unique MIDI Channel (1 - 16) and its unique Control Number (0 - 127). Therefore, the MIDI protocol allows the user to select between 2,048 different unique ”identifiers” (127 Control Numbers X 16 MIDI Channels) for each potentiometer of the MIDI Controller!

The MIDI Control Values (0 - 127) are used to change the values of the corresponding control of the connected MIDI device. If you imagine a volume knob on a synthesizer, 0 represents the lowest value (volume knob turned all the way down) and 127 represents the highest value (the volume knob turned all the way up).

4.2.4 LEDs

The following LEDs are used on the MIDI controller:

- (1) Edit Mode LED (red)
- (1) Preset 1 Edit LED (blue)
- (1) Preset 2 Edit LED (blue)
- (4) Potentiometer LEDs (green)

The single Edit Mode LED is activated when the user pushes the Edit Button. The LED turns on to signify that the user has entered EDIT MODE. The LED will continue to stay on until the user returns back to SESSION MODE.

The two Preset Edit LEDs are used to indicate which preset is being modified while in EDIT MODE. For example, if the user presses the Edit Button while using Preset 1, the Preset 1 Edit LED will illuminate and stay illuminated until the user returns back to SESSION MODE.

The 12 Potentiometer LEDs are used as a visual indication of the current potentiometer value. This is done using PWM (Pulse-Width-Modulation). Using this method, a value between 0-255 is used to control the brightness of each potentiometers associated LED.

4.3 5-Pin MIDI Output Circuit

The circuit below is the standard circuit used to send serial MIDI data over a standardized 5-pin MIDI cable.

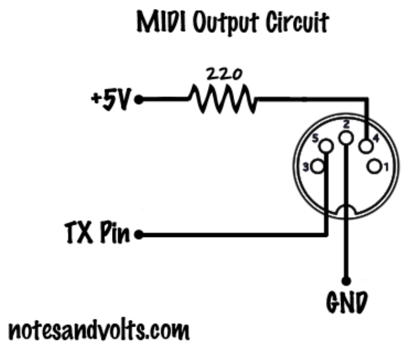


Figure 1: 5-pin MIDI output circuit

4.4 Finished Design

The circuit below is replicated on the next page for further detail.

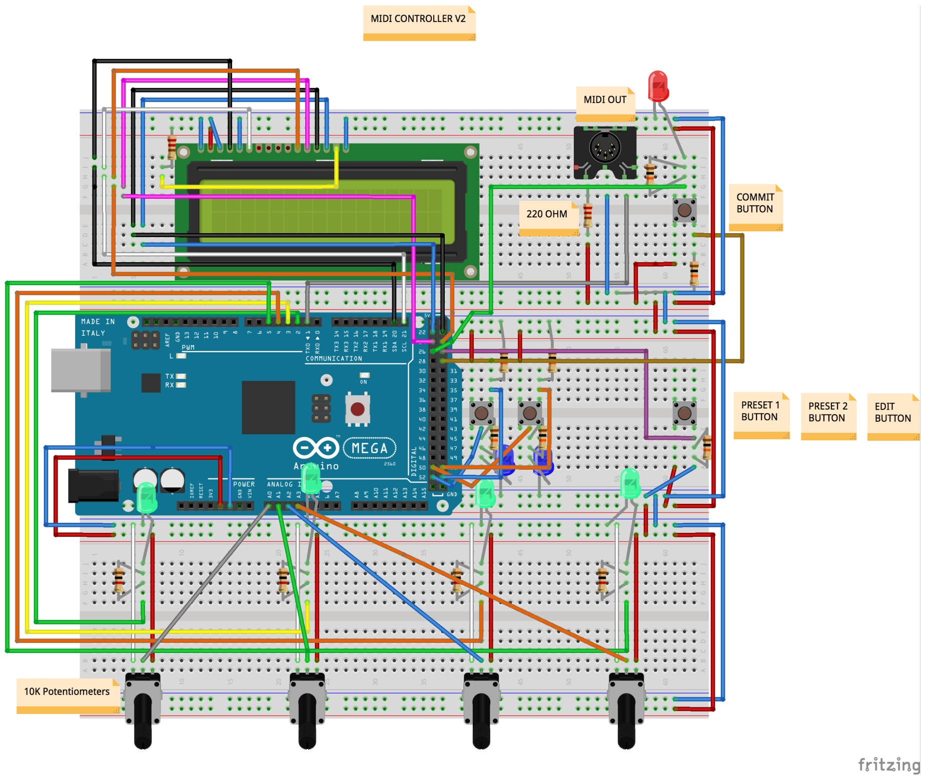
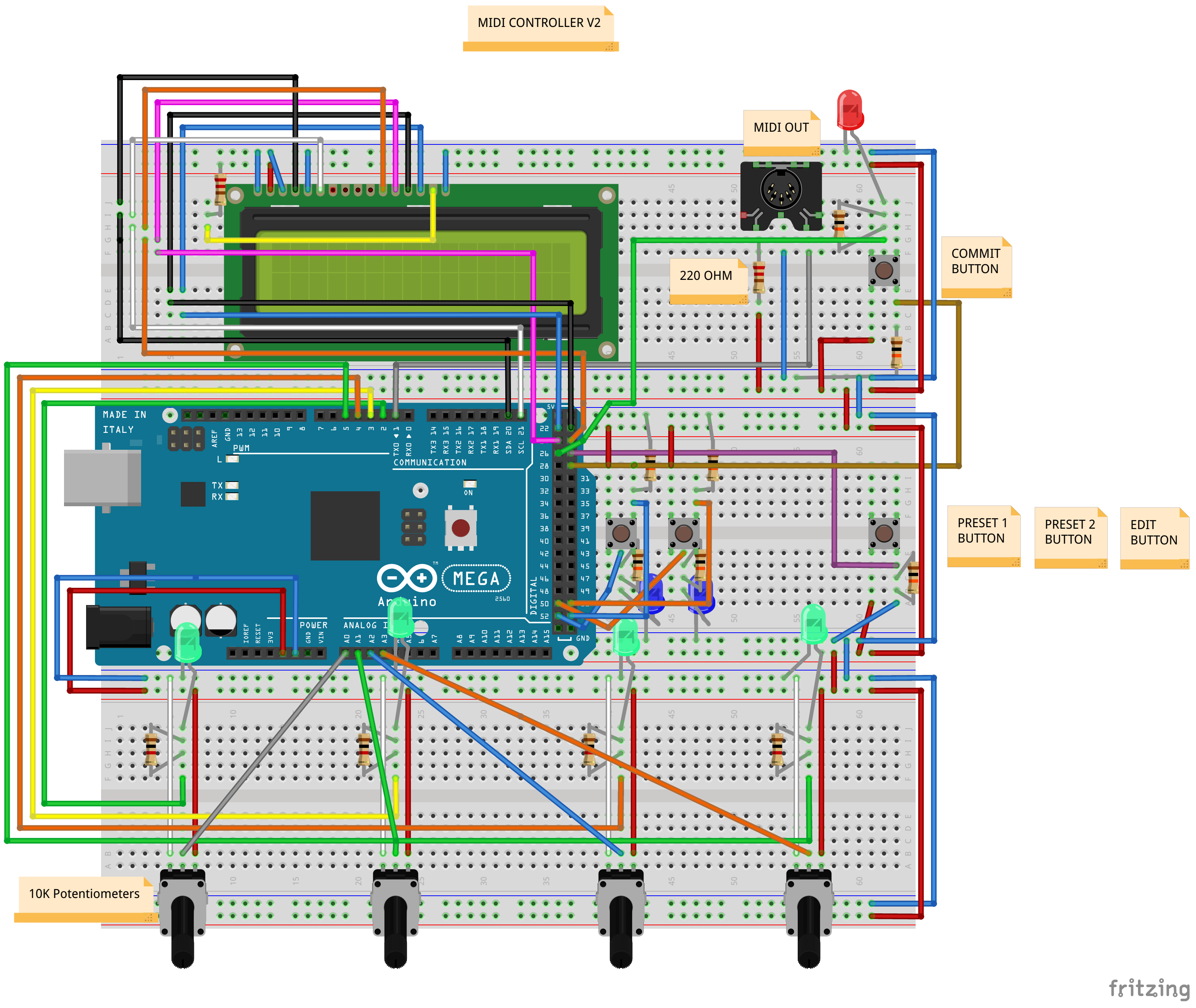


Figure 2: Completed Circuit Design


MIDI CONTROLLER V2
MIDI OUT
COMMIT BUTTON
220 OHM
PRESET 1 BUTTON
PRESET 2 BUTTON
EDIT BUTTON
10K Potentiometers
fritzing

5 Software

5.1 Software Design Philosophy

While developing the code, basic functionality was first implemented. This included LCD screen functionality, potentiometer and button connectivity, MIDI data parsing, LED Pulse-Width-Modulation control, EEPROM memory, serial communication, etc. Once the basic functionality was proven, the software was expanded into separate 'modules' that allowed the code to be more easily defined, debugged, and documented.

The Arduino (C++) code was broken into multiple '.ino' files. Arduino allows the code to be modularized into separate files as long as each file has the '.ino' extension and all are placed within the same parent folder (as shown below).

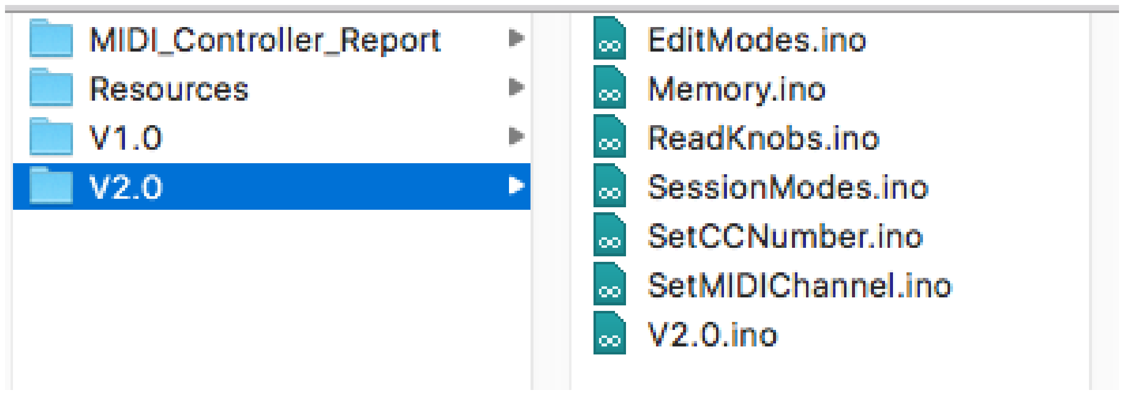


Figure 3: Modularized Arduino code

5.2 Code Modules

The following files were utilized as separate modules in the Arduino code:

- V2.ino
- Memory.ino
- SessionModes.ino
- ReadKnobs.ino
- EditModes.ino
- SetMIDIChannel.ino
- SetCCNumber.ino

5.2.1 V2.ino

The code in this module is considered the "main code" as it prepares the controller for use and continuously runs. It boots the controller into Preset 1 by default, accesses EEPROM variables for preset 1 and 2, opens the controller's serial communication ports, initializes the LCD display, sets output and input pins, and continuously runs Preset1Mode() method in its main loop.

- Includes Libraries
- Declares Variables
- Sets up variables and controller state
- Continuously runs MAIN function

5.2.2 Memory.ino

The code in this module reads the corresponding values previously stored into the EEPROM so that when the device is turned on, all of the preset values are first properly recalled (EEPROM memory is written in SetMIDIChannel and SetCCNumber modules).

5.2.3 SessionModes.ino

The code in this module is continuously being ran within the MAIN function of V2.0.ino and therefore continuously calls the ReadKnobs.ino methods (to send MIDI data) unless the user chooses to enter EDIT MODE. SessionModes.ino contains the Preset1Mode() and Preset2Mode() methods. The functionality of each of the methods is the same (the only difference is the preset each corresponds to). Each method first displays "SESSION 1 (or 2) MODE" onto the LCD. The methods then determine if the user wants to enter EDIT MODE by continuously reading the pin associated with the Edit Button and comparing the pin state to a flag. If the user wants to leave SESSION MODE to enter EDIT MODE, the Edit Button is pressed and the EditMode() methods are called from this module.

5.2.4 ReadKnobs.ino

The code in this module samples voltages read on the potentiometers to create MIDI messages and control the associated LEDs via PWM (Pule-Width-Modulation). The code continuously reads the pins associated with each of the potentiometers (k1 - k12) and maps the received analog range (0 Volts - 5Volts) to digital values (0-1023). This range is mapped to the PWM range (0-255) to control the brightness of each of the LEDs as well as the Control Value range (0 - 127) to control the external MIDI devices.

Each knob has its own unique MIDI message array. The code will obtain the value of the potentiometer that is being moved (by comparing the previous knob state to the current knob state) and append the current mapped Control Value to a MIDI Channel Number and a Control Number and place these three values into the knob's array. This array is then continuously sent over the serial port as long as the knob continues to move.

Because the chip on the Arduino micro controller is able to execute a large number of operations per second, the values of each of the knobs on the MIDI controller are sampled so fast that the transmission of data is sent smoothly with no observable "glitches" or latency!

5.2.5 EditModes.ino

The code in this module is used to access the MIDI Channel and Control Number editing processes. The code first displays "EDIT SESH 1 (or 2)" on the LCD. Next, the code toggles between MIDI Channel and Control Number editing by utilizing a 'state-detection' sub-circuit that calls either the SetMIDIChannelPreset() or SetCCNumberPreset() methods. The user simply pushes the Edit Mode Button to toggle between the two. Finally, the code determines if the user wants to stay in EDIT MODE or return to SESSION MODE by continuously reading the Preset 1 / Preset 2 button states. If the user wants to return to SESSION MODE, the Preset 1 / Preset 2 button is pushed.

5.2.6 SetMIDIChannel.ino

The code within this module allows the user to set and save the MIDI Channel of each knob while in EDIT MODE. The code first displays the words "EDIT CHANNEL" to the LCD to alert the user that MIDI Channel editing is occurring. The code then will obtain the value of the knob that is being moved (by comparing the previous knob state to the current knob state within a timeframe of 5ms) and display the selected MIDI Channel (1-16) to the LCD along with the knob number (k1 - k12). When the user pushes the Commit Button, the MIDI Channel of that knob is stored to its specific memory in the internal EEPROM of the Arduino. A message displaying the knob (k1 - k12) and its new MIDI Channel (1 - 16) is then written to the LCD to notify the user of a successful save.

5.2.7 SetCCNumber.ino

The code within this module allows the user to set and save the Control Number of each knob while in EDIT MODE. The code first displays the words "EDIT NUM" to the LCD to alert the user that Control Number editing is occurring. The code then will obtain the value of the knob that is being moved (by comparing the previous knob state to the current knob state within a timeframe of 5ms) and display the selected Control Number (0-127) to the LCD along with the knob number (k1 - k12). When the user pushes the Commit Button, the Control Number of that knob is stored to its specific memory in the internal EEPROM of the Arduino. A message displaying the knob (k1 - k12) and its new Control Number (0 - 127) is then written to the LCD to notify the user of a successful save.

References

[1] (2020, April). [Online]. Available: http://hinton-instruments.co.uk/reference/midi/protocol/index.htm

[2] (2020, April). [Online]. Available: https://www.midi.org/specifications-old/item/table-1-summary-of-midi-message

[3] (2020, April). [Online]. Available: https://www.arduino.cc/en/Tutorial/BuiltInExamples

[4] April 2020. [Online]. Available: https://www.notesandvolts.com/2015/03/midi-for-arduino-build-midi-output.html

[5] April 2020. [Online]. Available: https://learn.sparkfun.com/tutorials/midi-tutorial/all

6 Arduino Code

The complete Arduino source code is provided in the following pages:

```
/* Author: Austin Stockwell
   Date: 04/08/2020
   Description: This is the code of an Arduino
            MIDI Controller that sends MIDI control data
            to an external device via 5-PIN MIDI
            cable without the use of Arduino MIDI libraries.

            A user chooses between Session Mode and Edit
            Mode via a single pushbutton...

            In Session Mode, the user can continuously send
            MIDI to any device that recieves MIDI
            Control Messages.  The user does this by using
            a combination of the 4 knobs.  Each knob features
            a dedicated LED controlled by Pulse-Width Modulation (PWM).

            In Edit Mode, the user can configure the messages.
            Each knob can be configured to transmit any Control Number
            (0-127) on any MIDI Channel (1-16).  Saved values
            are stored to the board's internal EEPROM when a user
            pushes the "COMMIT" button.
*/
/*  ------------------------------------------------------- */
// INCLUDED LIBRARIES
/*  ------------------------------------------------------- */
#include <EEPROM.h>
#include <LiquidCrystal.h>

/*  ------------------------------------------------------- */
// DECLARED VARIABLES
/*  ------------------------------------------------------- */
/* LCD */
const int rs = 20, en = 21, d4 = 25, d5 = 24, d6 = 23, d7 = 22;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

/* LEDs */
const int knobLEDs[] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
const int editModeLED = 26;
const int seshLED = 52;
const int sesh2LED = 50;

/* KNOBS / KNOB STATES*/
const int k1 = A0;
const int k2 = A1;
const int k3 = A2;
const int k4 = A3;
// Prest 1
int Preset1KnobStates[4];
int resetKnobFlag = 1;
int k1state;
int k2state;
int k3state;
int k4state;
int k1OldState = 0;
int k2OldState = 0;
int k3OldState = 0;
int k4OldState = 0;
```

```
// Preset 2
int Preset2KnobStates[4];
int resetKnobFlag2 = 1;
int k1state2;
int k2state2;
int k3state2;
int k4state2;
int k1OldState2 = 0;
int k2OldState2 = 0;
int k3OldState2 = 0;
int k4OldState2 = 0;

/* BUTTONS / BUTTON STATES */
const int seshButton = 53;
const int sesh2Button = 51;
const int editButton = 27;
const int commitButton = 29;
int seshButtonState;
int sesh2ButtonState;
int editButtonState;
int commitButtonState;

/* DEFAULT VALUES */
// Preset 1
int k1newChannel = 176;
int k2newChannel = 176;
int k3newChannel = 176;
int k4newChannel = 176;
int k1newNumber = 0;
int k2newNumber = 1;
int k3newNumber = 2;
int k4newNumber = 3;
// Preset 2
int k1newChannel2 = 177;
int k2newChannel2 = 177;
int k3newChannel2 = 177;
int k4newChannel2 = 177;
int k1newNumber2 = 0;
int k2newNumber2 = 1;
int k3newNumber2 = 2;
int k4newNumber2 = 3;

/*  -------------------------------------------------------- */
// SETUP
/*  -------------------------------------------------------- */
void setup() {
 //Serial.begin(9600);
 Serial.begin(31250); // Used to communicate over MIDI cable
  Memory(); // Read in data from EEPROM

  /* LCD */
  lcd.begin(16, 2);

  /* LEDS */
  for (int index = 0; index < 4; index ++)
  {
    pinMode((knobLEDs[index]), OUTPUT);
  }
  pinMode(seshLED, OUTPUT);
  pinMode(sesh2LED, OUTPUT);
  pinMode(editModeLED, OUTPUT);

  /* KNOBS */
  pinMode(k1, INPUT);
  pinMode(k2, INPUT);
  pinMode(k3, INPUT);
  pinMode(k4, INPUT);

  /* BUTTONS */
  pinMode(seshButton, INPUT);
  pinMode(editButton, INPUT);
}

/*  ------------------------------------------------------- */
// MAIN
/*  ------------------------------------------------------- */
void loop() {
  Preset1Mode();
}
```

```
/* Author: Austin Stockwell
    Date: 04/08/2020
    Description: Memory()
      Access variables stored in memory for each preset
*/
void Memory(){
  // Preset 1
  k1newChannel = EEPROM.read(0);
  k2newChannel = EEPROM.read(1);
  k3newChannel = EEPROM.read(2);
  k4newChannel = EEPROM.read(3);
  k1newNumber = EEPROM.read(12);
  k2newNumber = EEPROM.read(13);
  k3newNumber = EEPROM.read(14);
  k4newNumber = EEPROM.read(15);

  // Preset 2
  k1newChannel2 = EEPROM.read(24);
  k2newChannel2 = EEPROM.read(25);
  k3newChannel2 = EEPROM.read(26);
  k4newChannel2 = EEPROM.read(27);
  k1newNumber2 = EEPROM.read(36);
  k2newNumber2 = EEPROM.read(37);
  k3newNumber2 = EEPROM.read(38);
  k4newNumber2 = EEPROM.read(39);
}
```

```
/* Author: Austin Stockwell
    Date: 04/08/2020
    Description: SessionMode()
      Ensure user wants to send MIDI data or switch to Edit Modes...
*/
void Preset1Mode() {
  Serial.println("SESSION MODE");

  // Setup LCD & LED for Session Mode
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("  SESSION MODE  ");
  digitalWrite(seshLED, LOW);
  digitalWrite(sesh2LED, LOW);
  digitalWrite(editModeLED, LOW);

  // Stay in Session MODE until user chooses another MODE
  int sessionFlag = 0;
  while (sessionFlag != 1) {
    editButtonState = digitalRead(editButton);
    sesh2ButtonState = digitalRead(sesh2Button);

    // Read Value and compose MIDI message for Preset 1...
    ReadKnobsPreset1();

    // IF user wants to switch to Preset 2...
    if(sesh2ButtonState == 1){
      // Pass last Preset 1 knob states...
      Preset1KnobStates[0] = k1OldState;
      Preset1KnobStates[1] = k2OldState;
      Preset1KnobStates[2] = k3OldState;
      Preset1KnobStates[3] = k4OldState;
      resetKnobFlag2 = 1; // Ensures Preset 2 values are not overwritten
      Preset2Mode();
    }

    // If user wants to enter Edit Mode...
    if(editButtonState == 1) {
      Serial.println("Leaving Session 1...");
      sessionFlag = 1;
    }
  }
  sessionFlag = 0; // Reset flag for next use of Session Mode
  EditMode1(); // Returns to Edit Mode
}

/****************************************************/
void Preset2Mode(){
  Serial.println("SESSION 2 MODE");

  // Setup LCD & LED for Session2 Mode
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print(" SESSION 2 MODE  ");
  digitalWrite(seshLED, LOW);
  digitalWrite(sesh2LED, LOW);
  digitalWrite(editModeLED, LOW);
  // Stay in Session 2 Mode until user chooses another MODE
  int session2Flag = 0;
  while (session2Flag != 1) {
    editButtonState = digitalRead(editButton);
    seshButtonState = digitalRead(seshButton);

    // Read value of knobs...
    ReadKnobsPreset2();

    // IF user wants to switch to Prest 1...
    if(seshButtonState == 1){
      // Save current state of Preset 2 knob states to pass back to
      // Preset 1 so Preset 1 values do not change to what they were set
      // to previously...
      Preset2KnobStates[0] = k1OldState2;
      Preset2KnobStates[1] = k2OldState2;
      Preset2KnobStates[2] = k3OldState2;
      Preset2KnobStates[3] = k4OldState2;
      resetKnobFlag = 1; // Ensures Preset 1 previous values are not overwritten
      Preset1Mode();
    }

    // If user wants to enter CC Edit Mode...
    if(editButtonState == 1) {
      Serial.println("Leaving Session 2...");
      session2Flag = 1;
    }
  }
  session2Flag = 0; // Reset flag for next use of Session Mode
  EditMode2(); // Returns to Edit Mode
}
```

```
/* Author: Austin Stockwell
    Date: 04/08/2020
    Description: ReadKnobs()
       Samples voltages read on potentiometers to create MIDI Messages & control knob LEDs
*/
void ReadKnobsPreset1(){
  Serial.println("READ KNOBS PRESET 1 ");

    // Array to holds each knobs CC CONTROL CHANGE VALUE
    int mappedValArray[4];

      Serial.println("Prest 2 last state");
      Serial.println(Preset2KnobStates[0]);
      Serial.println("This preset's last state");
      Serial.println(k1OldState);

      while(resetKnobFlag == 1){
         Serial.println("IN WHILE");

      /* When the user switches back to this preset (Preset 1), this circuit prevents the
         previous values of Preset 1 from being overwritten to the values of Preset 2. */
        if(k1OldState != Preset2KnobStates[0]){
          k1OldState = Preset2KnobStates[0];
          Serial.println("WE HAVE CHANGED K1OLDSTATE TO THE PRESET 2 VALUE");
          Serial.println(k1OldState);
        }
        if(k2OldState != Preset2KnobStates[1]){
          k2OldState = Preset2KnobStates[1];
        }
         if(k3OldState != Preset2KnobStates[2]){
          k3OldState = Preset2KnobStates[2];
        }
        if(k4OldState != Preset2KnobStates[3]){
          k4OldState = Preset2KnobStates[3];
        }
        break;
      }
        resetKnobFlag = 0; // Set flag to 0 so we only set prest 2 values once

         // Continuously read knob values to see if knob has been moved
          k1state = analogRead(k1);
          k2state = analogRead(k2);
          k3state = analogRead(k3);
          k4state = analogRead(k4);

          // Map recieved potentiomter values to control LED w/PWM (0-255)
          int k1mappedLED = map(k1state, 0, 1021, 0, 255);
          int k2mappedLED = map(k2state, 0, 1021, 0, 255);
          int k3mappedLED = map(k3state, 0, 1021, 0, 255);
          int k4mappedLED = map(k4state, 0, 1021, 0, 255);
          analogWrite(2, k1mappedLED);
          analogWrite(3, k2mappedLED);
          analogWrite(4, k3mappedLED);
          analogWrite(5, k4mappedLED);

            // IF knob1 moves send midi message
            if(abs (k1state - k1OldState) >= 5){
```

```
            Serial.println("KNOB HAS MOVED PRESET 1");

            // Re-read value and map to MIDI message format (0 - 127)
            k1state = analogRead(k1);
            mappedValArray[0] = map(k1state, 0, 1021, 0, 127);

            // SEND MIDI MESSAGE!!!
            byte k1MIDIMess[3] = {k1newChannel, k1newNumber, mappedValArray[0]};
            for (int k1index = 0; k1index < 3; k1index++)
            {
             Serial.write(k1MIDIMess[k1index]);
            }

            // Holds the most recent value of the knob for use in determining
            // if the knob has moved in the next iteration of ReadKnobsPreset1()
            k1OldState = k1state;
          }

          if (abs(k2state - k2OldState) >= 5){
            k2state = analogRead(k2);
            mappedValArray[1] = map(k2state, 0, 1021, 0, 127);

           byte k2MIDIMess[3] = {k2newChannel, k2newNumber, mappedValArray[1]};
           for (int k2index = 0; k2index < 3; k2index++)
           {
             Serial.write(k2MIDIMess[k2index]);
           }
            k2OldState = k2state;
          }

          if (abs(k3state - k3OldState) >= 5){
            k3state = analogRead(k3);
            mappedValArray[2] = map(k3state, 0, 1021, 0, 127);

           byte k3MIDIMess[3] = {k3newChannel, k3newNumber, mappedValArray[2]};
           for (int k3index = 0; k3index < 3; k3index++)
           {
             Serial.write(k3MIDIMess[k3index]);
           }
            k3OldState = k3state;
          }

          if (abs(k4state - k4OldState) >= 5){
            k4state = analogRead(k4);
            mappedValArray[3] = map(k4state, 0, 1021, 0, 127);

           byte k4MIDIMess[3] = {k4newChannel, k4newNumber, mappedValArray[3]};
           for (int k4index = 0; k4index < 3; k4index++)
           {
             Serial.write(k4MIDIMess[k4index]);
           }
            k4OldState = k4state;
          }
}

/****************************************************/
void ReadKnobsPreset2(){
```

```
Serial.println("READ KNOBS PRESET 2");

  // Array to holds each knobs CC CONTROL CHANGE VALUE
  int mappedValArray2[4];

    /* When the user switches back to this preset (Preset 2), this circuit prevents the
       previous values of Preset 2 from being overwritten to the values of Preset 1. */
    while(resetKnobFlag2 == 1){
      Serial.println("IN WHILE");
      // If the last state of Preset 2 DOES NOT equal the last state of Preset 1....
      // Set the last state of Preset 1 equal to last state of Preset 2...
      if(k1OldState2 != Preset1KnobStates[0]){
        Serial.println("WE HAVE CHANGED K1OLDSTATE2 TO THE PRESET 1 VALUE");
        k1OldState2 = Preset1KnobStates[0];
      }
      if(k2OldState2 != Preset1KnobStates[1]){
        k2OldState2 = Preset1KnobStates[1];
      }
       if(k3OldState2 != Preset1KnobStates[2]){
        k3OldState2 = Preset1KnobStates[2];
      }
      if(k4OldState2 != Preset1KnobStates[3]){
        k4OldState2 = Preset1KnobStates[3];
      }
      break;
    }
      resetKnobFlag2 = 0; // Set resetKnobFlag to 0 so we only reset the values of Preset 2
                      // to the values of Preset 1 one time (prevents data overwrite)...

 // Continuously read knob values to see if knob has been moved
  k1state2 = analogRead(k1);
  k2state2 = analogRead(k2);
  k3state2 = analogRead(k3);
  k4state2 = analogRead(k4);

  // Map recieved potentiomter values to control LED w/PWM (0-255)
  int k1mappedLED2 = map(k1state2, 0, 1021, 0, 255);
  int k2mappedLED2 = map(k2state2, 0, 1021, 0, 255);
  int k3mappedLED2 = map(k3state2, 0, 1021, 0, 255);
  int k4mappedLED2 = map(k4state2, 0, 1021, 0, 255);
  analogWrite(2, k1mappedLED2);
  analogWrite(3, k2mappedLED2);
  analogWrite(4, k3mappedLED2);
  analogWrite(5, k4mappedLED2);

    // IF knob1 moves send midi message
    if(abs (k1state2 - k1OldState2) >= 5){
       Serial.println("KNOB HAS MOVED PRESET 2");

        // Re-read value and map to MIDI message format (0 - 127)
        k1state2 = analogRead(k1);
        mappedValArray2[0] = map(k1state2, 0, 1021, 0, 127);

        // SEND MIDI MESSAGE!!!
        byte k1MIDIMess2[3] = {k1newChannel2, k1newNumber2, mappedValArray2[0]};
        for (int k1index2 = 0; k1index2 < 3; k1index2++)
        {
            Serial.write(k1MIDIMess2[k1index2]);
          }

          // Holds the most recent value of the knob for use in determining
          // if the knob has moved in the next iteration of ReadKnobsPreset1()
          k1OldState2 = k1state2;
      }

      if (abs(k2state2 - k2OldState2) >= 5){
          k2state2 = analogRead(k2);
          mappedValArray2[1] = map(k2state2, 0, 1021, 0, 127);

        byte k2MIDIMess2[3] = {k2newChannel2, k2newNumber2, mappedValArray2[1]};
        for (int k2index2 = 0; k2index2 < 3; k2index2++)
        {
            Serial.write(k2MIDIMess2[k2index2]);
        }
          k2OldState2 = k2state2;
      }

      if (abs(k3state2 - k3OldState2) >= 5){
          k3state2 = analogRead(k3);
          mappedValArray2[2] = map(k3state2, 0, 1021, 0, 127);

        byte k3MIDIMess2[3] = {k3newChannel2, k3newNumber2, mappedValArray2[2]};
        for (int k3index2 = 0; k3index2 < 3; k3index2++)
        {
            Serial.write(k3MIDIMess2[k3index2]);
        }
          k3OldState2 = k3state2;
      }

      if (abs(k4state2 - k4OldState2) >= 5){
          k4state2 = analogRead(k4);
          mappedValArray2[3] = map(k4state2, 0, 1021, 0, 127);

        byte k4MIDIMess2[3] = {k4newChannel2, k4newNumber2, mappedValArray2[3]};
        for (int k4index2 = 0; k4index2 < 3; k4index2++)
        {
            Serial.write(k4MIDIMess2[k4index2]);
        }
          k4OldState2 = k4state2;
      }
}
```

```
/* Author: Austin Stockwell
   Date: 04/08/2020
   Description: EditModes()
      Accesses the MIDI Channel / Control Number editing for each preset...
*/
void EditMode1(){
  Serial.println("PRESET 1 EDIT MODE");

  // Setup LCD & LEDs for Preset 2 Edit Mode
  lcd.clear();
  lcd.setCursor(0, 0); // (column, row)
  lcd.print("  EDIT SESH 1 ");
  digitalWrite(editModeLED, HIGH);
  digitalWrite(seshLED, HIGH);
  digitalWrite(sesh2LED, LOW);
  for (int index = 0; index < 4; index++) {
    digitalWrite(knobLEDs[index], LOW);
  }

  // Stay in Edit Mode until user presses Session 1 Button...
  int edit1Flag = 0;
  while (edit1Flag != 1) {
    seshButtonState = digitalRead(seshButton);
    editButtonState = digitalRead(editButton);

  // State detection sub-circuit toggles between MIDI CH /CC NUM editing
    int buttonPushCounter;   // Counter for the number of button presses
    int localbuttonState = 0;     // Current state of the button
    int lastButtonState = 0;     // Previous state of the button

        if (editButtonState != lastButtonState) {
          buttonPushCounter++;

          if (editButtonState = HIGH) {
            lastButtonState = 0;  // If its high, it was low before...
             delay(250);  // Debouncing of button presses for accuracy
          }
          if (editButtonState = LOW) {
            lastButtonState = 1;
            delay(250);
          }
            // SET MIDI CHANNEL
            if (buttonPushCounter % 2) {
              SetMIDIChannelPreset1();
            }
            // SET CONTROL NUMBER
            else{
              SetCCNumberPreset1();
            }
        }

    // If user wants to enter Session Mode...
    if (seshButtonState == 1){
      edit1Flag = 1;
    }
  } // while(edit2Flag != 1)
  edit1Flag = 0;  // Resets editFlag for next use
  digitalWrite(editModeLED, LOW); // Light off (leaving Edit Mode)
  Preset1Mode();  // Return to Session Mode
}

/****************************************************/
void EditMode2(){
  Serial.println("PRESET 2 EDIT MODE");

  // Setup LCD & LEDs for Preset 2 Edit Mode
  lcd.clear();
  lcd.setCursor(0, 0); // (column, row)
  lcd.print("  EDIT SESH 2 ");
  digitalWrite(editModeLED, HIGH);
  digitalWrite(sesh2LED, HIGH);
  digitalWrite(seshLED, LOW);
  for (int index = 0; index < 4; index++) {
    digitalWrite(knobLEDs[index], LOW);
  }

  // Stay in CCMode until user presses Session 2 Button
  int edit2Flag = 0;
  while (edit2Flag != 1) {
    sesh2ButtonState = digitalRead(sesh2Button);
    editButtonState = digitalRead(editButton);

  // State detection sub-circuit toggles between MIDI CH /CC NUM editing
    int buttonPushCounter;   // Counter for the number of button presses
    int localbuttonState = 0;     // Current state of the button
    int lastButtonState = 0;     // Previous state of the button

        if (editButtonState != lastButtonState) {
          buttonPushCounter++;

          if (editButtonState = HIGH) {
            lastButtonState = 0;  // If its high, it was low before...
             delay(250);  // Debouncing of button presses for accuracy
          }
          if (editButtonState = LOW) {
            lastButtonState = 1;
            delay(250);
          }
            // SET MIDI CHANNEL
            if (buttonPushCounter % 2) {
              SetMIDIChannelPreset2();
            }
            // SET CONTROL NUMBER
            else{
              SetCCNumberPreset2();
            }
        }
    // If user wants to enter Session 2 Mode...
    if (sesh2ButtonState == 1){
      edit2Flag = 1;
    }
  } // while(edit2Flag != 1)
  edit2Flag = 0;  // Resets editFlag for next use
  digitalWrite(editModeLED, LOW); // Light off (leaving Edit Mode)
  Preset2Mode();  // Return to Session Mode
}
```

```
/* Author: Austin Stockwell
    Date: 04/08/2020
    Description: SetMIDIChannel()
       Sets the MIDI Channel of each knob
*/
int SetMIDIChannelPreset1() {
  Serial.println("PRESET 1 CHANNEL EDITING");

  lcd.clear();
  lcd.setCursor(0, 0); // (column, row)
  lcd.print("  EDIT CHANNEL");
  digitalWrite(seshLED, HIGH);
  digitalWrite(sesh2LED, LOW);

 // Ensure user wants to continue editing MIDI Channel...
  int SetMIDIChannelPreset1Flag = 0;
  while (SetMIDIChannelPreset1Flag != 1) {
    seshButtonState = digitalRead(seshButton);
    editButtonState = digitalRead(editButton);

// KNOB 1
    // Get value from knob IF KNOB MOVES
    int oldk1state = analogRead(k1);
    delay(5);
    k1state = analogRead(k1);
    if(abs(oldk1state - k1state) > 8)
    {
      while(commitButtonState == 0 || editButtonState == 0)
      {
        commitButtonState = digitalRead(commitButton);
        editButtonState = digitalRead(editButton);

        // Continuously clear old value to write new value
        delay(100);  // Aids stability of LCD screen
        lcd.setCursor(8, 1);
        lcd.print("            ");
        lcd.setCursor(0, 1); // (column, row);
        lcd.print("MIDI CH: ");  // Continue to display "MIDI CH"

        // Read and write k1 value to LCD
        k1state = analogRead(k1);
        int k1mapChanDisp = map(k1state, 0, 1021, 1, 16);
        int k1mappedChannel = map(k1state, 0, 1021, 176, 191);
        lcd.setCursor(8, 1);
        lcd.print(k1mapChanDisp);

        if(editButtonState == 1){
          EditMode1();
          break;
        }

        if(commitButtonState == 1){
          // Write new channel value to newChannel var
          k1newChannel = k1mappedChannel;
          EEPROM.write(0,k1newChannel); // (address, val)
```

```
          // Dispaly new channel to LCD
```

```
        lcd.clear();
        lcd.setCursor(7,0);
        lcd.print("K1");
        lcd.setCursor(0,1);
        lcd.print("NEW CHANNEL: ");
        lcd.setCursor(12, 1); // (column, row);
        lcd.print(k1mapChanDisp);
        delay(2000);
        lcd.clear();

        // Continue to edit MIDI Channels...
        SetMIDIChannelPreset1();
        break;
      }
    } // WHILE editing MIDI Channel...
  } // IF knob was turned...

// KNOB 2
  int oldk2state = analogRead(k2);
  delay(5);
  k2state = analogRead(k2);
  if(abs(oldk2state - k2state) > 8)
  {
    while(commitButtonState == 0 || editButtonState == 0)
    {
      commitButtonState = digitalRead(commitButton);
      editButtonState = digitalRead(editButton);

      delay(100);  // Aids stability of LCD screen
      lcd.setCursor(8, 1);
      lcd.print("            ");
      lcd.setCursor(0, 1); // (column, row);
      lcd.print("MIDI CH: ");  // Continue to display "MIDI CH"

      k2state = analogRead(k2);
      int k2mapChanDisp = map(k2state, 0, 1021, 1, 16);
      int k2mappedChannel = map(k2state, 0, 1021, 176, 191);

      lcd.setCursor(8, 1);
      lcd.print(k2mapChanDisp);

      if(editButtonState == 1){
        EditMode1();
        break;
      }

      if(commitButtonState == 1){
         k2newChannel = k2mappedChannel;
         EEPROM.write(1,k2newChannel);

        lcd.clear();
        lcd.setCursor(7,0);
        lcd.print("K2");
        lcd.setCursor(0,1);
        lcd.print("NEW CHANNEL: ");
        lcd.setCursor(12, 1);
        lcd.print(k2mapChanDisp);
        delay(2000);
        lcd.clear();

        SetMIDIChannelPreset1();
        break;
      }
    }
  }

// KNOB 3
  int oldk3state = analogRead(k3);
  delay(5);
  k2state = analogRead(k3);
  if(abs(oldk3state - k3state) > 8)
  {
    while(commitButtonState == 0 || editButtonState == 0)
    {
      commitButtonState = digitalRead(commitButton);
      editButtonState = digitalRead(editButton);

      delay(100);
      lcd.setCursor(8, 1);
      lcd.print("            ");
      lcd.setCursor(0, 1);
      lcd.print("MIDI CH: ");

      k3state = analogRead(k3);
      int k3mapChanDisp = map(k3state, 0, 1021, 1, 16);
      int k3mappedChannel = map(k3state, 0, 1021, 176, 191);

      lcd.setCursor(8, 1);
      lcd.print(k3mapChanDisp);

      if(editButtonState == 1){
        EditMode1();
        break;
      }

      if(commitButtonState == 1){
         k3newChannel = k3mappedChannel;
         EEPROM.write(2,k3newChannel);

        lcd.clear();
        lcd.setCursor(7,0);
        lcd.print("K3");
        lcd.setCursor(0,1);
        lcd.print("NEW CHANNEL: ");
        lcd.setCursor(12, 1);
        lcd.print(k3mapChanDisp);
        delay(2000);
        lcd.clear();

        SetMIDIChannelPreset1();
        break;
      }
    }
  }
```

```
// KNOB 4
    int oldk4state = analogRead(k4);
    delay(5);
    k4state = analogRead(k4);
    if(abs(oldk4state - k4state) > 8)
    {
      while(commitButtonState == 0 || editButtonState == 0)
      {
        commitButtonState = digitalRead(commitButton);
        editButtonState = digitalRead(editButton);

        delay(100);
        lcd.setCursor(8, 1);
        lcd.print("            ");
        lcd.setCursor(0, 1);
        lcd.print("MIDI CH: ");

        k4state = analogRead(k4);
        int k4mapChanDisp = map(k4state, 0, 1021, 1, 16);
        int k4mappedChannel = map(k4state, 0, 1021, 176, 191);

        lcd.setCursor(8, 1);
        lcd.print(k4mapChanDisp);

        if(editButtonState == 1){
          EditMode1();
          break;
        }

        if(commitButtonState == 1){
          k4newChannel = k4mappedChannel;
          EEPROM.write(3,k4newChannel);

          lcd.clear();
          lcd.setCursor(7,0);
          lcd.print("K4");
          lcd.setCursor(0,1);
          lcd.print("NEW CHANNEL: ");
          lcd.setCursor(12, 1);
          lcd.print(k4mapChanDisp);
          delay(2000);
          lcd.clear();

          SetMIDIChannelPreset1();
          break;
        }
      }
    }

    // If user wants to continue in Edit Mode...
    if (editButtonState == 1) {
      SetMIDIChannelPreset1Flag == 1;
      delay(250); // Aids in button stability to return to Edit Mode...
      SetCCNumberPreset1();
      break;
    }
    // If user wants to return to Session Mode...
    if (seshButtonState == 1) {
      SetMIDIChannelPreset1Flag == 1;
      Preset2Mode();
      break;
    }
  } // while (SetMIDIChannelPreset2Flag !=1)
} // SetMIDIChannelPreset2() method

/****************************************************/
int SetMIDIChannelPreset2() {
  Serial.println("PRESET 2 CHANNEL EDITING");

  lcd.clear();
  lcd.setCursor(0, 0); // (column, row)
  lcd.print("  EDIT CHANNEL");

 // Ensure user wants to continue editing MIDI Channel...
  int setMIDIChannelPreset2Flag = 0;
  while (setMIDIChannelPreset2Flag != 1) {
    sesh2ButtonState = digitalRead(sesh2Button);
    editButtonState = digitalRead(editButton);

// KNOB 1
// Get value from knob IF KNOB MOVES
    int oldk1state2 = analogRead(k1);
    delay(5);
    k1state2 = analogRead(k1);
    if(abs(oldk1state2 - k1state2) > 8)
    {
      while(commitButtonState == 0 || editButtonState == 0)
      {
        commitButtonState = digitalRead(commitButton);
        editButtonState = digitalRead(editButton);

        // Continuously clear old value to write new value
        delay(100);  // Aids stability of LCD screen
        lcd.setCursor(8, 1);
        lcd.print("            ");
        lcd.setCursor(0, 1); // (column, row);
        lcd.print("MIDI CH: ");  // Continue to display "MIDI CH"

        // Read and write k1 value to LCD
        k1state2 = analogRead(k1);
        int k1mapChanDisp2 = map(k1state2, 0, 1021, 1, 16);
        int k1mappedChannel2 = map(k1state2, 0, 1021, 176, 191);
        lcd.setCursor(8, 1);
        lcd.print(k1mapChanDisp2);

        if(editButtonState == 1){
          EditMode2();
          break;
        }

        if(commitButtonState == 1){
          // Write new channel value to newChannel var
          k1newChannel2 = k1mappedChannel2;
        EEPROM.write(24,k1newChannel2); // (address, val)

        // Dispaly new channel to LCD
        lcd.clear();
        lcd.setCursor(7,0);
        lcd.print("K1");
        lcd.setCursor(0,1);
        lcd.print("NEW CHANNEL: ");
        lcd.setCursor(12, 1); // (column, row);
        lcd.print(k1mapChanDisp2);
        delay(2000);
        lcd.clear();

        // Continue to edit MIDI Channels...
        SetMIDIChannelPreset2();
        break;
      }
    } // WHILE editing MIDI Channel...
  } // IF knob was turned...

// KNOB 2
  int oldk2state2 = analogRead(k2);
  delay(5);
  k2state2 = analogRead(k2);
  if(abs(oldk2state2 - k2state2) > 8)
  {
    while(commitButtonState == 0 || editButtonState == 0)
    {
      commitButtonState = digitalRead(commitButton);
      editButtonState = digitalRead(editButton);

      delay(100);  // Aids stability of LCD screen
      lcd.setCursor(8, 1);
      lcd.print("            ");
      lcd.setCursor(0, 1); // (column, row);
      lcd.print("MIDI CH: ");  // Continue to display "MIDI CH"

      k2state2 = analogRead(k2);
      int k2mapChanDisp2 = map(k2state2, 0, 1021, 1, 16);
      int k2mappedChannel2 = map(k2state2, 0, 1021, 176, 191);

      lcd.setCursor(8, 1);
      lcd.print(k2mapChanDisp2);

      if(editButtonState == 1){
        EditMode2();
        break;
      }

      if(commitButtonState == 1){
         k2newChannel2 = k2mappedChannel2;
         EEPROM.write(25,k2newChannel2);

        lcd.clear();
        lcd.setCursor(7,0);
        lcd.print("K2");
        lcd.setCursor(0,1);
        lcd.print("NEW CHANNEL: ");
        lcd.setCursor(12, 1);
        lcd.print(k2mapChanDisp2);
        delay(2000);
        lcd.clear();

        SetMIDIChannelPreset2();
        break;
      }
    }
  }

// KNOB 3
  int oldk3state2 = analogRead(k3);
  delay(5);
  k2state2 = analogRead(k3);
  if(abs(oldk3state2 - k3state2) > 8)
  {
    while(commitButtonState == 0 || editButtonState == 0)
    {
      commitButtonState = digitalRead(commitButton);
      editButtonState = digitalRead(editButton);

      delay(100);
      lcd.setCursor(8, 1);
      lcd.print("            ");
      lcd.setCursor(0, 1);
      lcd.print("MIDI CH: ");

      k3state2 = analogRead(k3);
      int k3mapChanDisp2 = map(k3state2, 0, 1021, 1, 16);
      int k3mappedChannel2 = map(k3state2, 0, 1021, 176, 191);

      lcd.setCursor(8, 1);
      lcd.print(k3mapChanDisp2);

      if(editButtonState == 1){
        EditMode2();
        break;
      }

      if(commitButtonState == 1){
         k3newChannel2 = k3mappedChannel2;
         EEPROM.write(26,k3newChannel2);

        lcd.clear();
        lcd.setCursor(7,0);
        lcd.print("K3");
        lcd.setCursor(0,1);
        lcd.print("NEW CHANNEL: ");
        lcd.setCursor(12, 1);
        lcd.print(k3mapChanDisp2);
        delay(2000);
        lcd.clear();

        SetMIDIChannelPreset2();
        break;
        }
      }
    }

// KNOB 4
    int oldk4state2 = analogRead(k4);
    delay(5);
    k4state2 = analogRead(k4);
    if(abs(oldk4state2 - k4state2) > 8)
    {
      while(commitButtonState == 0 || editButtonState == 0)
      {
        commitButtonState = digitalRead(commitButton);
        editButtonState = digitalRead(editButton);

        delay(100);
        lcd.setCursor(8, 1);
        lcd.print("            ");
        lcd.setCursor(0, 1);
        lcd.print("MIDI CH: ");

        k4state2 = analogRead(k4);
        int k4mapChanDisp2 = map(k4state2, 0, 1021, 1, 16);
        int k4mappedChannel2 = map(k4state2, 0, 1021, 176, 191);

        lcd.setCursor(8, 1);
        lcd.print(k4mapChanDisp2);

        if(editButtonState == 1){
          EditMode2();
          break;
        }

        if(commitButtonState == 1){
          k4newChannel2 = k4mappedChannel2;
          EEPROM.write(27,k4newChannel2);

          lcd.clear();
          lcd.setCursor(7,0);
          lcd.print("K4");
          lcd.setCursor(0,1);
          lcd.print("NEW CHANNEL: ");
          lcd.setCursor(12, 1);
          lcd.print(k4mapChanDisp2);
          delay(2000);
          lcd.clear();

          SetMIDIChannelPreset2();
          break;
        }
      }
    }

    // If user wants to continue in Edit Mode...
    if (editButtonState == 1) {
      setMIDIChannelPreset2Flag == 1;
      delay(250); // Aids in button stability to return to Edit Mode...
      SetCCNumberPreset2();
      break;
    }
    // If user wants to return to Session Mode...
    if (sesh2ButtonState == 1) {
      setMIDIChannelPreset2Flag == 1;
      Preset2Mode();
      break;
    }
  } // while (setMIDIChannelFlag !=1)
} // SetMIDIChannel() method
```

```
/* Author: Austin Stockwell
   Date: 04/08/2020
   Description: SetCCNumber()
     Sets the Control Number of each knob
*/
int SetCCNumberPreset1(){
  Serial.println("PRESET 1 CONTROL NUMBER EDITING");

  lcd.clear();
  lcd.setCursor(0, 0); // (column, row)
  lcd.print("    EDIT NUM   ");

  // Ensure user wants to continue editing CC Number...
  int setCCNumberPreset1Flag = 0;
  while (setCCNumberPreset1Flag != 1) {
    seshButtonState = digitalRead(seshButton);
    editButtonState = digitalRead(editButton);

// KNOB 1
        int oldk1state = analogRead(k1);
        delay(5);
        k1state = analogRead(k1);
        if(abs(oldk1state - k1state) > 8)
        {
          while(commitButtonState == 0 || editButtonState == 0)
          {
            commitButtonState = digitalRead(commitButton);
            editButtonState = digitalRead(editButton);

            delay(100);  // Aids stability of LCD screen
            lcd.setCursor(8, 1);
            lcd.print("            ");
            lcd.setCursor(0, 1); // (column, row);
            lcd.print("CC NUM: ");

            // Read and write k1 value to LCD
            k1state = analogRead(k1);
            int k1mappedNumber = map(k1state, 0, 1021, 0, 127);
            lcd.setCursor(8, 1);
            lcd.print(k1mappedNumber);

            if(editButtonState == 1){
              EditMode1();
              break;
            }

            if(commitButtonState == 1){
              // Write new channel value to newChannel var
              k1newNumber = k1mappedNumber;
              EEPROM.write(12,k1newNumber); // (address,val)

              // Dispaly new channel to LCD
              lcd.clear();
              lcd.setCursor(7,0);
              lcd.print("K1");
              lcd.setCursor(0,1);
              lcd.print("NEW cc NUM:");
              lcd.setCursor(11, 1); // (column, row);
              lcd.print(k1newNumber);
              delay(2000); // Display message for 2 seconds to confirm...
              lcd.clear();

              // Continue to edit MIDI Channels...
              SetCCNumberPreset1();
            }
          } // WHILE editing CC Number...
        } // IF knob was turned...

// KNOB 2
        int oldk2state = analogRead(k2);
        delay(5);
        k2state = analogRead(k2);
        if(abs(oldk2state - k2state) > 8)
        {
          while(commitButtonState == 0 || editButtonState == 0)
          {
            commitButtonState = digitalRead(commitButton);
            editButtonState = digitalRead(editButton);

            delay(100);
            lcd.setCursor(8, 1);
            lcd.print("        ");
            lcd.setCursor(0, 1);
            lcd.print("CC NUM: ");

            k2state = analogRead(k2);
            int k2mappedNumber = map(k2state, 0, 1021, 0, 127);
            lcd.setCursor(8, 1);
            lcd.print(k2mappedNumber);

            if(editButtonState == 1){
              EditMode1();
              break;
            }

            if(commitButtonState == 1){
              k2newNumber = k2mappedNumber;
              EEPROM.write(13,k2newNumber);

              lcd.clear();
              lcd.setCursor(7,0);
              lcd.print("K2");
              lcd.setCursor(0,1);
              lcd.print("NEW cc NUM:");
              lcd.setCursor(11, 1);
              lcd.print(k2newNumber);
              delay(2000);
              lcd.clear();

              SetCCNumberPreset1();
              break;
            }
          }
        }
```

```
// KNOB 3
        int oldk3state = analogRead(k3);
        delay(5);
        k3state = analogRead(k3);
        if(abs(oldk3state - k3state) > 8)
        {
          while(commitButtonState == 0 || editButtonState == 0)
          {
            commitButtonState = digitalRead(commitButton);
            editButtonState = digitalRead(editButton);

            delay(100);
            lcd.setCursor(8, 1);
            lcd.print("            ");
            lcd.setCursor(0, 1);
            lcd.print("CC NUM: ");

            k3state = analogRead(k3);
            int k3mappedNumber = map(k3state, 0, 1021, 0, 127);
            lcd.setCursor(8, 1);
            lcd.print(k3mappedNumber);

            if(editButtonState == 1){
              EditMode1();
              break;
            }

            if(commitButtonState == 1){
              k3newNumber = k3mappedNumber;
              EEPROM.write(14, k3newNumber);

              lcd.clear();
              lcd.setCursor(7,0);
              lcd.print("K3");
              lcd.setCursor(0,1);
              lcd.print("NEW cc NUM:");
              lcd.setCursor(11, 1);
              lcd.print(k3newNumber);
              delay(2000);
              lcd.clear();

              SetCCNumberPreset1();
              break;
            }
          }
        }

// KNOB 4
        int oldk4state = analogRead(k4);
        delay(5);
        k4state = analogRead(k4);
        if(abs(oldk4state - k4state) > 8)
        {
          while(commitButtonState == 0 || editButtonState == 0)
          {
            commitButtonState = digitalRead(commitButton);
          editButtonState = digitalRead(editButton);

          delay(100);
          lcd.setCursor(8, 1);
          lcd.print("            ");
          lcd.setCursor(0, 1);
          lcd.print("CC NUM: ");

          k4state = analogRead(k4);
          int k4mappedNumber = map(k4state, 0, 1021, 0, 127);
          lcd.setCursor(8, 1);
          lcd.print(k4mappedNumber);

          if(editButtonState == 1){
            EditMode1();
            break;
          }

          if(commitButtonState == 1){
            k4newNumber = k4mappedNumber;
            EEPROM.write(15, k4newNumber);

            lcd.clear();
            lcd.setCursor(7,0);
            lcd.print("K4");
            lcd.setCursor(0,1);
            lcd.print("NEW cc NUM:");
            lcd.setCursor(11, 1);
            lcd.print(k4newNumber);
            delay(2000);
            lcd.clear();

            SetCCNumberPreset1();
            break;
          }
        }
      }

    // If user wants to continue in Edit Mode...
    if (editButtonState == 1) {
      setCCNumberPreset1Flag = 1;
      delay(250); // Aids in button stability to return to Edit Mode...
      SetMIDIChannelPreset1();
      break;
    }
    // If user wants to return to Session Mode...
    if (seshButtonState == 1) {
      setCCNumberPreset1Flag = 1;
      Preset2Mode();
      break;
    }
  } // while(SetCCNumberPreset2Flag != 1)
} // SetCCNumberPreset2() method

/*****************************************************/
int SetCCNumberPreset2(){
  Serial.println("PRESET 2 CONTROL NUMBER EDITING");
```

```
 // Edit / save Control Number of knobs of Preset 2
  lcd.clear();
  lcd.setCursor(0, 0); // (column, row)
  lcd.print("    EDIT NUM   ");

  // Ensure user wants to continue editing CC Number...
  int setCCNumberPreset2Flag = 0;
  while (setCCNumberPreset2Flag != 1) {
    sesh2ButtonState = digitalRead(sesh2Button);
    editButtonState = digitalRead(editButton);

// KNOB 1
        int oldk1state2 = analogRead(k1);
        delay(5);
        k1state2 = analogRead(k1);
        if(abs(oldk1state2 - k1state2) > 8)
        {
          while(commitButtonState == 0 || editButtonState == 0)
          {
            commitButtonState = digitalRead(commitButton);
            editButtonState = digitalRead(editButton);

            delay(100);  // Aids stability of LCD screen
            lcd.setCursor(8, 1);
            lcd.print("            ");
            lcd.setCursor(0, 1); // (column, row);
            lcd.print("CC NUM: ");

            // Read and write k1 value to LCD
            k1state2 = analogRead(k1);
            int k1mappedNumber2 = map(k1state2, 0, 1021, 0, 127);
            lcd.setCursor(8, 1);
            lcd.print(k1mappedNumber2);

            if(editButtonState == 1){
              EditMode2();
              break;
            }

            if(commitButtonState == 1){
              // Write new channel value to newChannel var
              k1newNumber2 = k1mappedNumber2;
              EEPROM.write(36,k1newNumber2); // (address,val)

              // Dispaly new channel to LCD
              lcd.clear();
              lcd.setCursor(7,0);
              lcd.print("K1");
              lcd.setCursor(0,1);
              lcd.print("NEW cc NUM:");
              lcd.setCursor(11, 1); // (column, row);
              lcd.print(k1newNumber2);
              delay(2000); // Display message for 2 seconds to confirm...
              lcd.clear();
```

```
              // Continue to edit MIDI Channels...
```

```
            SetCCNumberPreset2();
          }
        } // WHILE editing CC Number...
      } // IF knob was turned...

// KNOB2
      int oldk2state2 = analogRead(k2);
      delay(5);
      k2state2 = analogRead(k2);
      if(abs(oldk2state2 - k2state2) > 8)
      {
        while(commitButtonState == 0 || editButtonState == 0)
        {
          commitButtonState = digitalRead(commitButton);
          editButtonState = digitalRead(editButton);

          delay(100);
          lcd.setCursor(8, 1);
          lcd.print("            ");
          lcd.setCursor(0, 1);
          lcd.print("CC NUM: ");

          k2state2 = analogRead(k2);
          int k2mappedNumber2 = map(k2state2, 0, 1021, 0, 127);
          lcd.setCursor(8, 1);
          lcd.print(k2mappedNumber2);

          if(editButtonState == 1){
            EditMode2();
            break;
          }

          if(commitButtonState == 1){
            k2newNumber2 = k2mappedNumber2;
            EEPROM.write(37,k2newNumber2);

            lcd.clear();
            lcd.setCursor(7,0);
            lcd.print("K2");
            lcd.setCursor(0,1);
            lcd.print("NEW cc NUM:");
            lcd.setCursor(11, 1);
            lcd.print(k2newNumber2);
            delay(2000);
            lcd.clear();

            SetCCNumberPreset2();
            break;
          }
        }
      }

//KNOB 3
      int oldk3state2 = analogRead(k3);
      delay(5);
      k3state2 = analogRead(k3);
      if(abs(oldk3state2 - k3state2) > 8)
```

```
        {
          while(commitButtonState == 0 || editButtonState == 0)
          {
            commitButtonState = digitalRead(commitButton);
            editButtonState = digitalRead(editButton);

            delay(100);
            lcd.setCursor(8, 1);
            lcd.print("            ");
            lcd.setCursor(0, 1);
            lcd.print("CC NUM: ");

            k3state2 = analogRead(k3);
            int k3mappedNumber2 = map(k3state2, 0, 1021, 0, 127);
            lcd.setCursor(8, 1);
            lcd.print(k3mappedNumber2);

            if(editButtonState == 1){
              EditMode2();
              break;
            }

            if(commitButtonState == 1){
              k3newNumber2 = k3mappedNumber2;
              EEPROM.write(38, k3newNumber2);

              lcd.clear();
              lcd.setCursor(7,0);
              lcd.print("K3");
              lcd.setCursor(0,1);
              lcd.print("NEW cc NUM:");
              lcd.setCursor(11, 1);
              lcd.print(k3newNumber2);
              delay(2000);
              lcd.clear();

              SetCCNumberPreset2();
              break;
            }
          }
        }

// KNOB 4
        int oldk4state2 = analogRead(k4);
        delay(5);
        k4state2 = analogRead(k4);
        if(abs(oldk4state2 - k4state2) > 8)
        {
          while(commitButtonState == 0 || editButtonState == 0)
          {
            commitButtonState = digitalRead(commitButton);
            editButtonState = digitalRead(editButton);

            delay(100);
            lcd.setCursor(8, 1);
            lcd.print("            ");
            lcd.setCursor(0, 1);
          lcd.print("CC NUM: ");

          k4state2 = analogRead(k4);
          int k4mappedNumber2 = map(k4state2, 0, 1021, 0, 127);
          lcd.setCursor(8, 1);
          lcd.print(k4mappedNumber2);

          if(editButtonState == 1){
            EditMode2();
            break;
          }

          if(commitButtonState == 1){
            k4newNumber2 = k4mappedNumber2;
            EEPROM.write(39, k4newNumber2);

            lcd.clear();
            lcd.setCursor(7,0);
            lcd.print("K4");
            lcd.setCursor(0,1);
            lcd.print("NEW cc NUM:");
            lcd.setCursor(11, 1);
            lcd.print(k4newNumber2);
            delay(2000);
            lcd.clear();

            SetCCNumberPreset2();
            break;
          }
        }
      }

    // If user wants to continue in Edit Mode...
    if (editButtonState == 1) {
      setCCNumberPreset2Flag = 1;
      delay(250); // Aids in button stability to return to Edit Mode...
      SetMIDIChannelPreset2();
      break;
    }
    // If user wants to return to Session Mode...
    if (sesh2ButtonState == 1) {
      setCCNumberPreset2Flag = 1;
      Preset2Mode();
      break;
    }
  } // while(SetCCNumberPreset2Flag != 1)
} // SetCCNumberPreset2() method
```