MIDI Bass Pedal

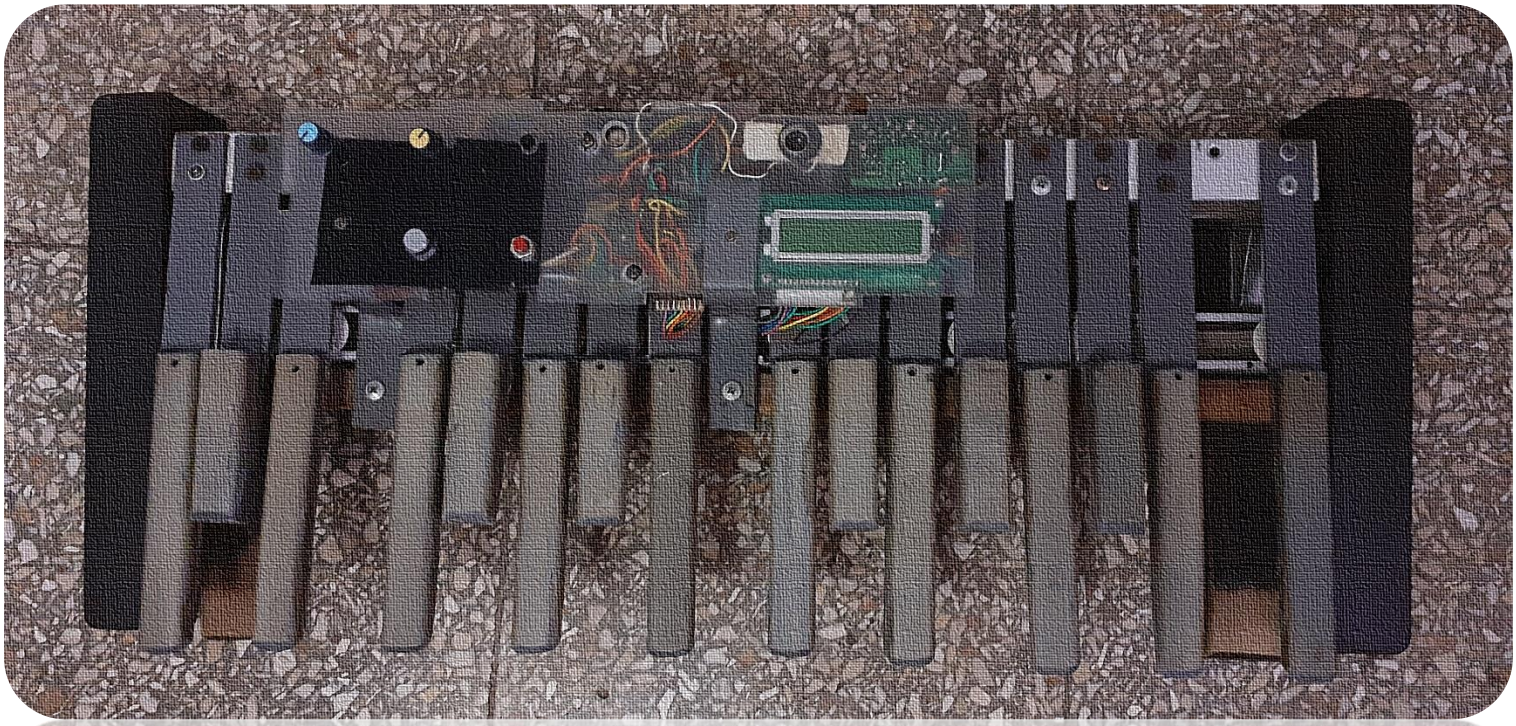


Figure 1 > Working Prototype

Author: Wilfredo Massó Turro

This PDF provides the MIDI Bass Pedal (shown above) documentation. Schematic, code and procedures are explained here.

Table of Contents

4 **Control Panel**
4 Control Panel Overview
4 Panel controls description
6 **Schematic**
6 List of parts
9 **The Arduino code**
9 Arduino code overview
13 PROCEDURES
14 **Proteus simulation**
15 **Adding quick presets to notes**
16 **Conclusion**

Control Panel

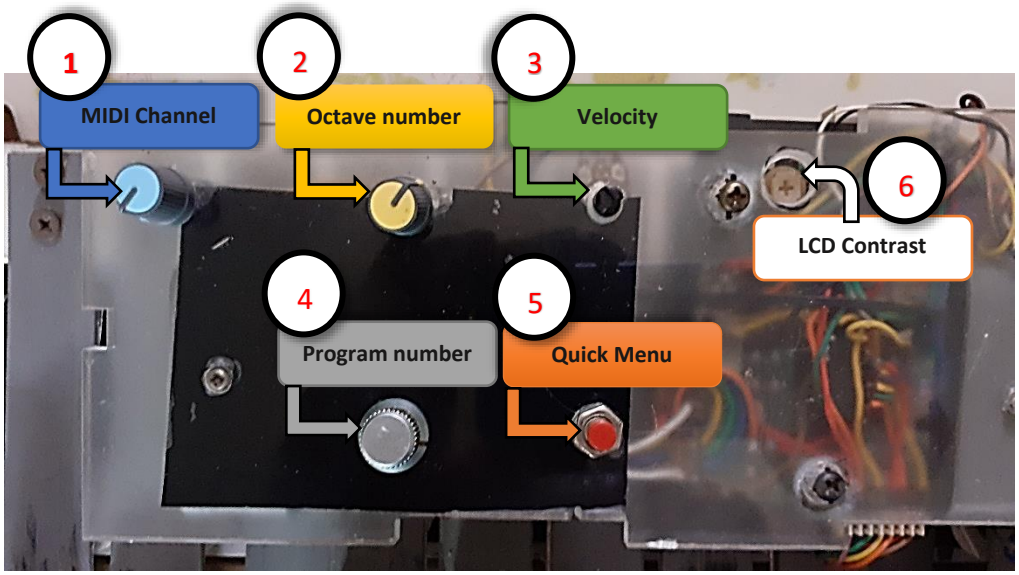
Home

The **Control Panel** board allows you to make adjustments to the MIDI Bass Pedal.

For actions such as tweaking the controls you can change things like select different Programs, change MIDI channels, change octave number and velocity as well. The momentary pushbutton let you access to a quick menu mode where you can change many parameters at once just holding this button down and then moving a control or pressing a keyboard note (previously programmed in the code). In this mode you can also activate a second voice with different settings.

Control Panel Overview

Home

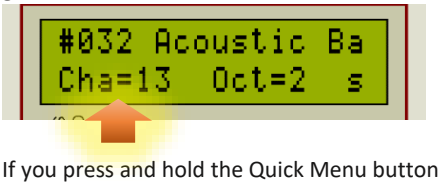


Panel controls description

Home

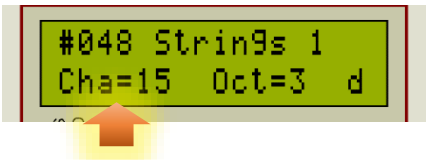
MIDI Channel

This control allows you to assign any of the 16 available MIDI channels for communicate with the tone generator.



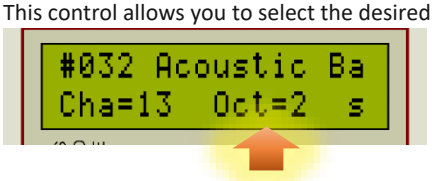
1

If you press and hold the Quick Menu button then you can change the MIDI channel number assigned to the second voice.



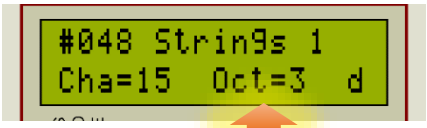
Octave number

This control allows you to select the desired octave number for the Pedal notes you play.



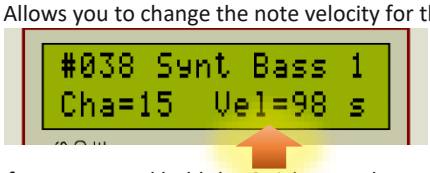
2

If you press and hold the Quick Menu button then you can change the octave number assigned to the second voice.



Velocity

Allows you to change the note velocity for the pedal notes you play.

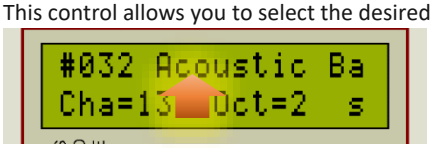


3

If you press and hold the Quick Menu button then you can change the note velocity assigned to the second voice.

Program number

This control allows you to select the desired sound (Program) for the pedal notes you play.



4

If you press and hold the Quick Menu button then you can change the Program number assigned to the second voice.

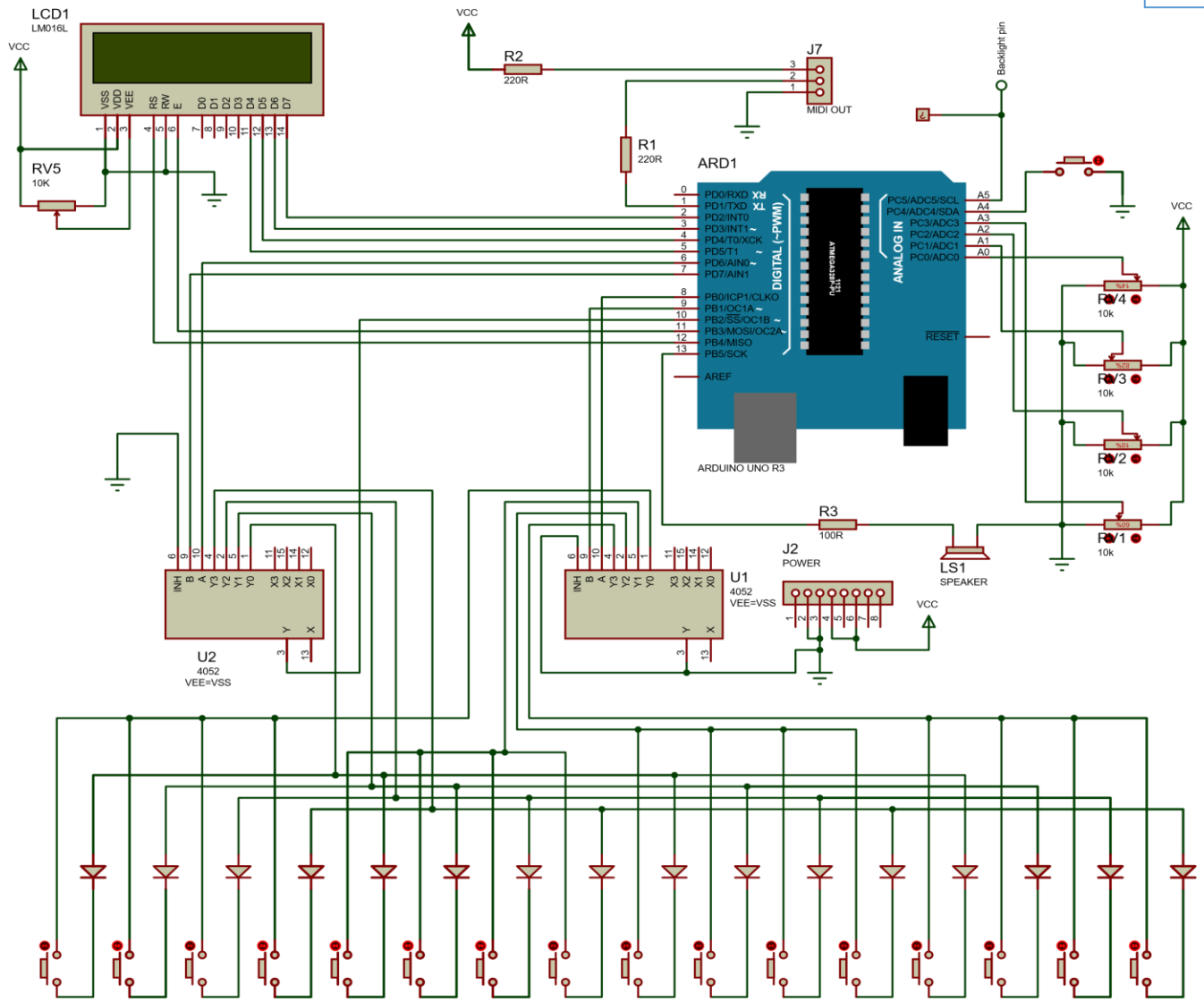
Quick Menu

5

This button allow you perform quick settings in conjunction with controls or pedal notes. While button pressed, the pedal notes will not produce any sound. In this mode the pedal notes act as preset buttons and must be already programmed in the code.

Schematic

Home



NOTE

The schematic was designed and simulated in Proteus 8.8. Code programmed in Arduino IDE 1.8.9. The (.hex) and (.elf) files used in Proteus simulation were generated in Arduino Pro IDE.

Home

	List of parts	Quantity	Comment
1-	Arduino Uno Board	1	
2-	10 k linear potentiometers	4	Try good quality ones
3-	10 k preset potentiometer	1	Adjust LCD contrast
4-	1n814 diode	16	
5-	CD 4052	2	Dual 1 of 4 analog multiplexer demultiplexer
6-	220-ohms resistor	2	
7-	HD 44780 compatible LCD	1	
8-	Momentary pushbutton	1	
9-	Small 8-ohms speaker	1	
10-	100-ohms resistor	1	

The Arduino code

Home

This is my first Arduino project. I used some examples codes that I found in a book and others in Instructables. With the time my code was growing and even going better because I spent a lot of time learning about the Arduino board and its programming techniques. Delays were just used in the welcome screen part of the code but anymore. I am working in the next version of the pedal where a jog encoder will be used in place of potentiometers.

Want to remark that I don't used debounce code (but soft switches in the keyboard layout) or MIDI libraries in the project and hope this fact doesn't constituted an unfortunate error. Anyway, the pedal prototype is working how expected. Video links: Performing with the Pedal, LCD and controls.

Arduino code overview

Arduino code

```
/*
  MIDI Bass Pedal

  Author: Wilfredo Massó Turro

  Date: 9/3/2019

  A MIDI Pedal with LCD display and auto on/off backlight.
  Two voices and a quick preset mode, let you to assign predefined
  parameters just pressing & holding a momentary button and a keyboard note.

*/

/*
  The circuit:
  LCD RS pin to digital pin 12
  LCD Enable pin to digital pin 11
  LCD D4 pin to digital pin 5
  LCD D5 pin to digital pin 4
  LCD D6 pin to digital pin 3
  LCD D7 pin to digital pin 2
  LCD R/W pin to ground
  10K potentiometer:
  ends to +5V and ground
  wiper to LCD VO pin (pin 3)
  10K potentiometer on pin A0
  10K potentiometer on pin A1
  10K potentiometer on pin A2
  10K potentiometer on pin A3
  momentary button switch on pin A4 with pullup resistors activated
  CD 4052       Dual 1 of 4 decoder (see MIDI Bass Pedal schematic)
*/

// include the library code:
#include <LiquidCrystal.h>
#include "GMpatches.h"

// initialize the library by associating any needed LCD interface pin
// with the Arduino pin number it is connected to
const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
LiquidCrystal lcd(rs, en, d4, d5, d6, d7);

// Generally, you should use "unsigned long" for variables that hold time
// The value will quickly become too large for an int to store
```

Home

```
unsigned long previousMillis = 0;  // will store last time L.C.D. was scrolled
unsigned long displayMillis = 0;
unsigned long displayLight = 0;

// constants won't change:
const long interval = 500; //used for timed interrupts

// pins where the control sensors are attached
const int sensorProgram = A0;
const int sensorChannel = A1;
const int sensorVelocity = A2;
const int sensorOctave = A3;

//Strings - variables storing text for show on the LCD
String stringOct;
String stringChn;
String stringVel;
String stringPgm;

// variables will change
int count;                  //for scolling L.C.D.
int sensorMin = 1023;       // minimum sensor value
int sensorMax = 0;          // maximum sensor value
int pgmSelect, chnSelect, octSelect, veloSelect, pgmSelect2, chnSelect2, octSelect2, veloSelect2;
byte sensors[] {sensorProgram, sensorChannel, sensorVelocity, sensorOctave};
byte sensorsValues[] {0, 0 , 0, 0};
byte lastSensorsValues [] {0, 0, 0, 0};
byte scaleDiv[] {16, 16, 16, 64};
int sentByte = -1; //this value ensures sending a Program Change after power on
//corresponding with the potentiometer position
int sentByte2 = -1;
int r0, r1, r2, r3;
int s0 = 6, s1 = 7 , s2 = 8, s3 = 9, inPin = 10, prgMixPin = A4;
byte currentNote[16] {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
byte lastNote[16] {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};

// Bank Select message matrix
byte bankSel[] = {0xB0, 0x00, 0x00 , 0xB0, 0x20, 0x00};

// Program names matrix. put the 16 Programs you´ll need here
//in the order they must appear (see the GMPatches.h file for list)
char* progName[] = { GM032, GM033, GM134, GM035, GM036, GM037, GM038, GM039,
                     GM048, GM049, GM036, GM035, GM054, GM033, GM032, GM039
                   };
byte toggle = 1;

void setup() {
  //multiplexer out pins
  pinMode(s0, OUTPUT);
  pinMode(s1, OUTPUT);
  pinMode(s2, OUTPUT);
  pinMode(s3, OUTPUT);
  //backlight power pin
  pinMode(A5, OUTPUT);
  //multiplexer in pin
  pinMode(inPin, INPUT_PULLUP);
  pinMode(prgMixPin, INPUT_PULLUP);//quick menu button pin
  // initialize LCD and set up the number of columns and rows:

  lcd.begin(16 , 2);
  Serial.begin(31250);  //set to M.I.D.I.  baud rate
  lcd.home();
  digitalWrite(A5, HIGH);// turn backlight on
  printScreen("Arduino MIDI" , 2, 0);
  printScreen( "Bass Pedal" , 3, 1);
  delay(1500);
```

Home

```
 while (millis() < 2500) {
 lcd.scrollDisplayLeft();
   delay(100);
 } lcd.clear();
 delay(100);
 printScreen( "Version 1.0" , 3, 0);
 printScreen( "Revision #2" , 3, 1);
 // srolling LCD during two seconds
 delay(2000); //end of welcome screen
 //reset display
 lcd.clear();
}

void loop() {

 unsigned long currentMillis = millis();
 stringOct = ("Oct=");
 stringChn = ("Cha=");
 stringVel = ("Vel=");
 byte prgMix = digitalRead(prgMixPin);

 //sensors reading and LCD code ********************************************************************************+
 for (int y = 0; y < sizeof(sensors); y++) {
  int f = analogRead(sensors[y]);

  // apply the mapping to the sensor reading
  f = map(f, sensorMin, sensorMax, 255, 0);

  // in case the sensor value is outside the range seen during calibration
  f = constrain(f, 0, 256);

  sensorsValues[y] = f / scaleDiv[y];

  if ( toggle == 1) {

   pgmSelect = (sensorsValues[0] );
   pgmSelect2 = pgmSelect;
   chnSelect = (sensorsValues[1]  + 1) ;
   chnSelect2 = chnSelect ;
   veloSelect = (sensorsValues[2]  * 4 + 70)  ;
   veloSelect2 = veloSelect ;
   octSelect = (sensorsValues[3]) ;
   octSelect2 = octSelect ;
   toggle  = 0;
  }

  if  (sensorsValues[y] < lastSensorsValues[y] ||
     sensorsValues[y] > lastSensorsValues[y]) {

   digitalWrite(A5, HIGH); // turn on backlight
   displayLight = currentMillis;

   lastSensorsValues[y] = sensorsValues[y]; //ensures LCD scroll until any potentiometer is moving again

   switch (y) {
    case 0:
     if (prgMix == 0 ) {
      pgmSelect2 = (sensorsValues[0] );
      printScreen( progName[pgmSelect2] , 0, 0);

 }
      else {
       pgmSelect = (sensorsValues[0] );
       printScreen( progName[pgmSelect] , 0, 0);
      }
      count = 0;  // this will reset the LCD´s text position if it was previously scrolled
```

Home

```
    break;
   case 1:
    panic();
    if (prgMix == 0 ) {
     chnSelect2 = (sensorsValues[1] + 1);
     printScreen((stringChn + chnSelect2 + " " ) , 0, 1);
     sentByte2 = -1;
    }
    else {
     chnSelect = (sensorsValues[1] + 1);
     printScreen((stringChn + chnSelect + " " ) , 0, 1);
     sentByte = -1;
    }

    break;

   case 2:
    if (prgMix == 0 ) {
     veloSelect2 = (sensorsValues[2]  * 4 + 70)  ;
     printScreen((stringVel + veloSelect2 + " "), 8, 1);
    }
    else {
     veloSelect = (sensorsValues[2]  * 4 + 70)  ;
     printScreen((stringVel + veloSelect + " "), 8, 1);
    }
    displayMillis = currentMillis;
    beep();
    break;

   case 3:
    panic();
    if (prgMix == 0 ) {
     octSelect2 = (sensorsValues[3]) ;
     printScreen((stringOct + octSelect2 + "  "), 8, 1);
    }
    else {
     octSelect = (sensorsValues[3]) ;
     printScreen((stringOct + octSelect + "  "), 8, 1);
    }
    beep();
    break;
  }
 }

 // send characters to the LCD
 if (prgMix == 1) {
  if (count < 5)
   printScreen( progName[pgmSelect], 0, 0);
  printScreen(stringChn + chnSelect + " " , 0, 1);

  if (currentMillis - displayMillis <= interval * 4)
   printScreen((stringVel + veloSelect + " "), 8, 1);
  else
   printScreen(stringOct + octSelect + "  ", 8, 1);
 }

 if (prgMix == 0) {
  displayLight  = currentMillis;
  digitalWrite(A5, HIGH);

printScreen( progName[pgmSelect2], 0, 0);
  printScreen(stringChn + chnSelect2 + " " , 0, 1);

  if (currentMillis - displayMillis <= interval * 4)
   printScreen((stringVel + veloSelect2 + " "), 8, 1);
  else
```

Home

```
printScreen(stringOct + octSelect2 + "  ", 8, 1);
 count = 0;
  }
  if (chnSelect2 < 3) {
    lcd.print("s");
  }
  else if (chnSelect == chnSelect2)
    lcd.print("@");
  else lcd.print("d");
 }
 if (currentMillis - displayLight >= interval * 10)
  digitalWrite(A5, LOW);// turn off backlight, comment if you want backlight always on
 //*************************************************************************


 //multiplexer code
 //*************************************************************************
 for ( int val = 0; val < 16 ; val++) {

  r0 = bitRead(val, 0);
  r1 = bitRead(val, 1);
  r2 = bitRead(val, 2);
  r3 = bitRead(val, 3);

  digitalWrite(s0, r0);
  digitalWrite(s1, r1);
  digitalWrite(s2, r2);
  digitalWrite(s3, r3);
  //delay(2);
  currentNote[val] = digitalRead(inPin);

  switch (prgMix) {

   case 1:    // play mode

    if (currentNote[val] != lastNote[val]) { //a note event occurred
     lastNote[val] = currentNote[val];
     if (currentNote[val] == 0) {         //at least one key is pressed
      noteSend(0X90 | chnSelect - 1, 0x15 + (octSelect * 12) + val , veloSelect);
      if (chnSelect2 > 2 )
       noteSend(0X90 | chnSelect2 - 1, 0x15 + (octSelect2 * 12) + val , veloSelect2);
     }

     else {
      noteSend(0x80 | chnSelect - 1, 0x15 + (octSelect * 12) + val, veloSelect);
      if (chnSelect2 > 2)
       noteSend(0X80 | chnSelect2 - 1, 0x15 + (octSelect2 * 12) + val , veloSelect2);
     }
    }
    break;

   case 0: //menu mode

    if (currentNote[val] != lastNote[val]) { //a note event occurred
     lastNote[val] = currentNote[val];
     if (currentNote[val] == 0) {
      displayMillis = currentMillis;
      switch (val) {
       case 0:
        chnSelect2 = 5;
        octSelect2 = 2;
        pgmSelect2 = 9;
        sentByte2 = -1;
        break;
       case 1:
        chnSelect2 = 2;
```

Home

```
            beep();
            break;
            case 2:
              chnSelect2 = 5;
              octSelect2 = 2;
              pgmSelect2 = 12;
              sentByte2 = -1;
              break;
            case 3:
              beep();
              break;
            case 4:
              break;
            case 5:
              break;
            case 6:
              break;
            case 7:
              break;
            case 8:
              break;
            case 9:
              break;
            case 10:
              break;

          }
        }
        else {
          noteSend(0x80 | chnSelect - 1, 0x15 + (octSelect * 12) + val, veloSelect);
          if (chnSelect2 > 2)
            noteSend(0X80 | chnSelect2 - 1, 0x15 + (octSelect2 * 12) + val , veloSelect2);
        }
      }
    }
  }
  //*****************************************************************************

  if (currentMillis - previousMillis >= interval && count < 14 ) {
    // save the last time you scroll the L.C.D.
    previousMillis = currentMillis;
    count++;

    // Scrolling the LCD

    if ( count > 10) {
      stringPgm = (progName[pgmSelect]);
      String sub2 = stringPgm.substring(count - 9);
      lcd.clear();
      printScreen(sub2 , 0, 0);//hide numbers and show the full Program names
    }
  }

  //call a method
  if (sentByte != pgmSelect) {
    bnkSelect(progName[pgmSelect], chnSelect - 1);
    sentByte = pgmSelect;
  }
  if (sentByte2 != pgmSelect2) {

  bnkSelect(progName[pgmSelect2], chnSelect2 - 1);
    sentByte2 = pgmSelect2;
  }
}
```

```
// print text strings onto the LCD
void printScreen ( String text , int charN, int line) {
 // set the cursor to the top left
 lcd.setCursor(charN, line);
 lcd.print(text);
}

void beep () {
 tone(13, 1800, 20);
}

void noteSend(int command, int note, int velocity) {

 Serial.write(command);//send note on or note off command
 Serial.write(note);//send pitch data
 Serial.write(velocity);//send velocity data
}

//send the Bank Sel and Program Change message
void bnkSelect( String pgm, byte chn ) {
 for (int i = 0; i < sizeof(bankSel); i++) {
  Serial.write(bankSel[i]);
 }
 // extract and send the numbers of the Program name
 String sub = pgm.substring(1, 4) ;
 Serial.write(0xC0 | chn);
 Serial.write(sub.toInt());
 beep(); //end of message
}

void panic() {                    // All notes off
 for (int m = 0; m < 16; m++) {
  noteSend(0x80 | chnSelect - 1, 0x15 + (octSelect * 12) + m, 0x00);
  noteSend(0x80 | chnSelect2 - 1, 0x15 + (octSelect2 * 12) + m, 0x00);
 }
}
```

Home

PROCEDURE

Home

You can tweak some fragments of the code to suite your own needs. The example code above admits 16 MIDI channels, 16 Programs, 4 octaves and velocities from 70 to 127.
For example, if you need more than 16 sounds, do the following:

1- Change the first number in this declaration:

`byte scaleDiv[] {16, 16, 16, 64};` The first number in this matrix represent a divide by variable for Program control values.

Every control has a minimum value of 0 and a maximum value of 256 after calibrating, so if you need just 16 values for a full turn of the control then you must divide 256 by a specific number to obtain the desired result. The following code fragment shows how to do it:

```
 f = constrain(f, 0, 256);
 sensorsValues[y] = f / scaleDiv[y];
```

The first variable into the sensorValues[y] matrix correspond to a result number from dividing Program control value (ranges from 0 to 256) and the first variable into the scaleDiv[y] (16 in this case). Certainly, if you divide 256 by 16 you obtain a value of 16 representing the numbers of programs you can use. Remember that matrix´s indexes starts with number 0, so practical values ranges from 0 to 15 and not from 1 to 16.

Finally, if you change the first number in `byte scaleDiv[] {4, 16, 16, 64};` then 256/4 = 64.

In this way you have now 64 values in the Program Change control. What follow is to put the numbers of the programs for each control value:

```
// Program names matrix. put the 64 Programs you´ll need here
//in the order they must appear (see the GMPatches.h file for list)
char* progName[] = { GM032, GM033, GM134, GM035, GM036, GM037, GM038, GM039,
          GM048, GM049, GM036, GM035, GM054, GM033, GM032, GM039, x, x, x, x, x, x, x, x, , , , , , , , , , and so on until 64. };
```

Note: x represents any of the definitions in the GMPatches.h file.

NOTE

It´s very important that you follow the convention rules while writing the Programs numbers and names in the GMPatches.h file. Here are some examples:

#define GM032 "#032 Acoustic Bass"

Info: When you use this in the code, you are referencing the string between "".

#define GM001 "#001 Acoustic Piano"

The # character followed by a three places number without spaces. Info: Program change message uses this number and LCD routine uses the name after.

#define GM127 "#127 Anything Else"

Put one space character here.

Note: The General MIDI list of patches used in this document and hence in the pedal, are specific of the CASIO WK-1800 keyboard. Further, you can adapt this list (GMPatches.h file) to suite your MIDI generator patches list.

Proteus simulation

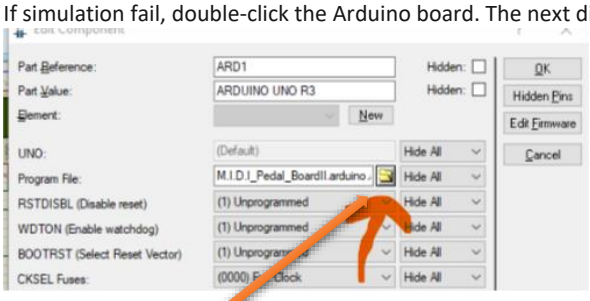
Home

If you are a Proteus 8.8 or 8.9 software user you can simulate the pedal. Just follow the next steps.

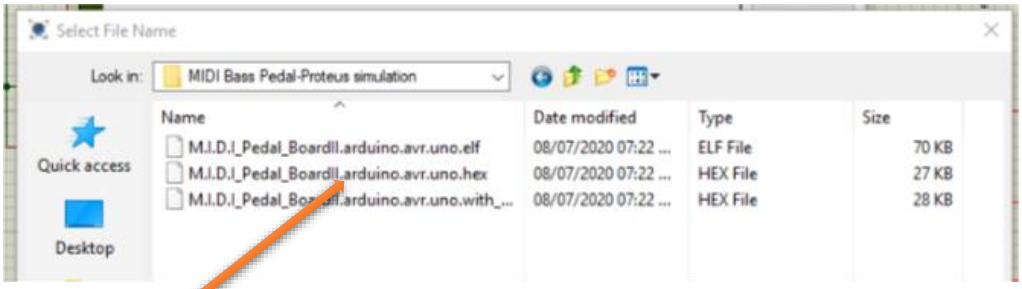
SIMULATION PROCEDURE

- In the File Explorer, navigate to the folder MIDI Bass Pedal-Proteus simulation located inside this Git folder and open it.
- Double-click the MIDI Bass Pedal Simulation file or click this link MIDI Bass Pedal Simulation.
  - Wait until Proteus is fully opened.
  - Run the simulation.

- If simulation fail, double-click the Arduino board. The next dialog will appear.

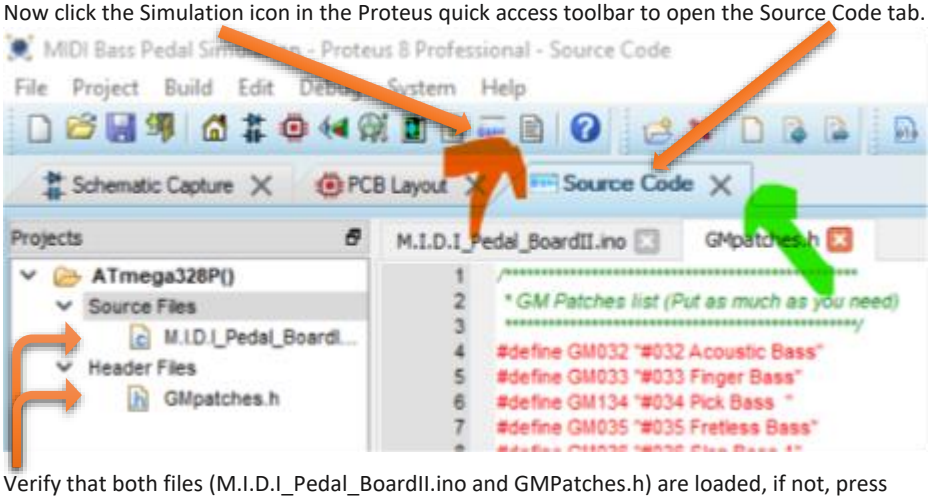


- Click in folder icon.



- Select the M.I.D.I_Pedal_BoardII.arduino.avr.uno.hex file and click the Open button.

- Now click the Simulation icon in the Proteus quick access toolbar to open the Source Code tab.



- Verify that both files (M.I.D.I_Pedal_BoardII.ino and GMPatches.h) are loaded, if not, press CTRL+ALT+A, navigate to folder M.I.D.I_Pedal_BoardII and add the files.
- Open the Build menu and click in Build Project or press CTRL+F7. Wait until finish the process.
- Go to the Schematic Capture tab and run the simulation or press F12.
- *Tip: Click in the red circle to hold the momentary switch pressed while simulating.*
- 

Adding quick presets to notes

Home

You can add quick presets to keyboard notes and uses it in conjunction with the Quick Menu button.
This is a programming trick and consequently the code must be uploaded to the Arduino board after changes.

PROCEDURE

Let, take a look at this fragment of the code.

```
case 0: //menu mode

  if (currentNote[val] != lastNote[val]) { //a note event occurred
    lastNote[val] = currentNote[val];
    if (currentNote[val] == 0) {
      displayMillis = currentMillis;
      switch (val) {
        case 0:
          chnSelect2 = 5;       //MIDI channel for the second voice
          octSelect2 = 2;       //Octave number for the second voice
          pgmSelect2 = 9;       //Program number for the second vice
          sentByte2 = -1;       //Instruct the program for send the values above
          break;
        case 1:
          chnSelect2 = 2;       //The second voice will sound only if MIDI channel > 2. Here no MIDI messages are sending for the second voice

          beep();
          break;
        case 2:
          chnSelect2 = 5;       // MIDI channel > 2. Here MIDI messages are sending for the second voice
          octSelect2 = 2;
          pgmSelect2 = 12;      //Program number 12 for the second voice
          sentByte2 = -1;
          break;
```

The Quick Menu button is hold pressed

The note number 0 is pressed

The note number 1 is pressed

The note number 2 is pressed

```
case 3:
 beep();
 break;
```

NOTE

The same procedure is valid for both voices.

Parameter	Voice 1	Voice2
Program number	pgmSelect	pgmSelect2
MIDI channel number	chnSelect	chnSelect2
Octave number	octSelect	octSelect2
Velocity	veloSelect	veloSelect2
***helper**	sentByte=-1	sentByte2=-1

- *helper: *You must put these sentences as is in order the program work properly. If you don't, the corresponding quick menu values will be not settled.*

Home

MIDI Bass Pedal specifications

Keyboard notes	**16**
Octaves	**4**
Power	**5VDC** (Arduino USB Port) **12VDC** (Arduino Power connector)
Simultaneous voices	**2**
Backlight	**Yes** (auto on/off)
Velocity	**Adjustable** (from 70 to 127)
Weight	
Dimensions	

Conclusion

The prototype pedal in which this document has been based was built with parts taken from old equipment. Everything, including keys, panel, PCB, were made from scrap and all the connections with wire links.

Perhaps you encounter grammatical or orthographic errors in this document. That is because my english language skills are very basic, and I did it without any help. Please be free of correct them or just let me know. Any comments are welcome.

Also let me know if you replicate this pedal or if this project was useful for you for any other purpose.

Bibliography

Arduino MIDI Bass Pedal by rczarnik in CircuitsArduino.

Send and receiving MIDI with Arduino by amandaghassaei in CircuitsAudio.