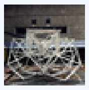
Send and Receive MIDI with Arduino

by **amandaghassaei** on June 9, 2012

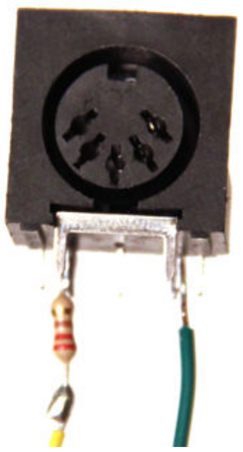
Table of Contents


Author:**amandaghassaei** amandaghassaei.com
Currently working for instructables!

Intro: Send and Receive MIDI with Arduino

This instructable will show you how to use an Arduino to send and receive a variety of MIDI messages so you can start building your own MIDI controllers and instruments. First I'll talk a little bit about MIDI protocol, if you're just looking for sample code skip ahead to steps 5-9.

If you know absolutely nothing about MIDI note, velocity, and pitchbend or are confused about what MIDI does and why you would want to use it, check out my What is MIDI? instructable.



Step 1: Bytes and Bits

To understand MIDI communication, you have to understand a little about bytes and bits. A byte is a packet of data used to store information. In MIDI protocol, each byte is made up of 8 bits; bits can only equal to 0 or 1. A sample byte is given below:

11010111

Each 1 or 0 in this byte is a bit. The leftmost bit is called the most significant bit (or MSB) and the rightmost bit is called the least significant bit (or LSB).

Bytes of the form above are binary numbers because they are expressed using only 1's and 0's. We can convert this number to base ten as well:

11010111 in binary (base 2) = 215 in decimal (base 10)

If you need help converting numbers from binary to decimal or vice versa check out Wolfram Alpha . Type in a binary number followed with "from binary to decimal" to get the decimal equivalent. Wolfram Alpha is also great for converting to and from hexadecimal .

Wikipedia is a good resource for more information about bytes and binary .

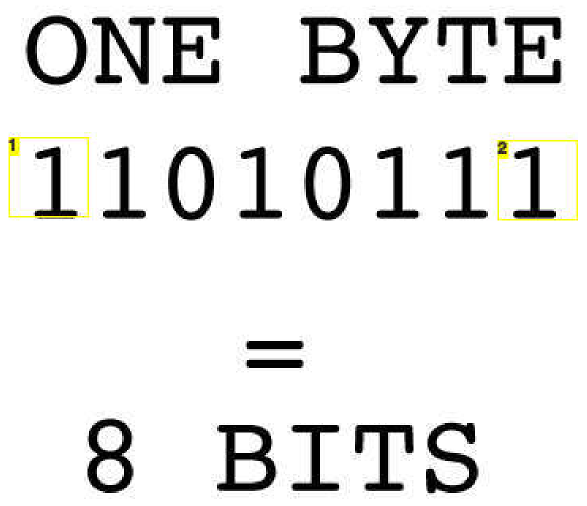


Image Notes
1. most significant bit (MSB)
2. least significant bit (LSB)

Step 2: A Bit About MIDI Protocol

A really basic overview of MIDI terms and concepts is given here .

MIDI messages are comprised of two components: commands and data bytes. The command byte tells the MIDI instrument what type of message is being sent and the subsequent data byte(s) store the actual data. For example a command byte might tell a MIDI instrument that it going to send information about pitchbend, and the data byte describes how much pitchbend and which note to pitchbend.

MIDI data bytes range from 0 to 127. Convert these numbers to binary and we see they range from 00000000 to 01111111, the important thing to notice here is that they always start with a 0 as the most significant bit (MSB). MIDI command bytes range from 128 to 255, or 1000000 to 11111111 in binary. Unlike data bytes, MIDI command bytes always start with a 1 as the MSB. This MSB is how a MIDI instrument differentiates between a command byte and a data byte.

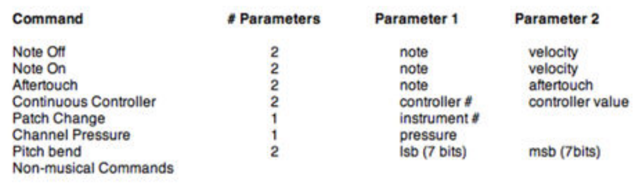
MIDI commands are further broken down by the following system:

The first half of the MIDI command byte (the three bits following the MSB) sets the type of command. More info about the meaning on each of these commands is here .
10000000 = note off
10010000 = note on
10100000 = aftertouch
10110000 = continuous controller
11000000 = patch change
11010000 = channel pressure
11100000 = pitch bend
11110000 = non-musical commands

The last half of the command byte sets the MIDI channel. All the bytes listed above would be in channel 0, command bytes ending in 0001 would be for MIDI channel 1, and so on.

All MIDI messages start with a command byte, some messages contain one data byte, others contain two or more (see image above). For example, a note on command byte is followed by two data bytes: note and velocity.

I'm going to explain how to use note on, note off, velocity, and pitchbend in this instructable, since these are the most commonly used commands. I'm sure you will be able to infer how to set up the others by the end of this.

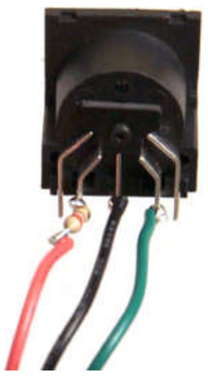


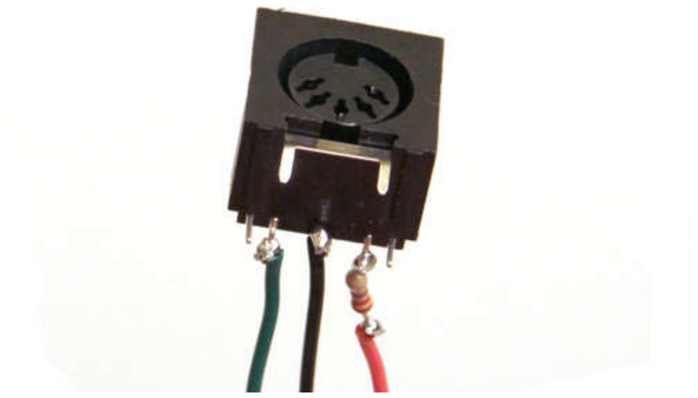
Command	# Parameters	Parameter 1	Parameter 2
Note Off	2	note	velocity
Note On	2	note	velocity
Aftertouch	2	note	aftertouch
Continuous Controller	2	controller #	controller value
Patch Change	1	instrument #	
Channel Pressure	1	pressure	
Pitch bend	2	lsb (7 bits)	msb (7bits)
Non-musical Commands			

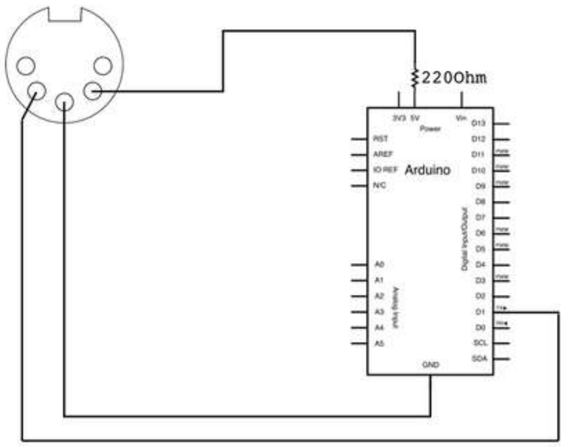
Step 3: Send MIDI Messages with Arduino- Hardware

Parts List:
MIDI connector Digikey CP-2350-ND
220Ohm 1/4watt resistor Digikey CF14JT220RCT-ND

Following the schematic above, solder a 220Ohm resistor to MIDI pin 4. Connect ground to MIDI pin 2 and 5V to MIDI pin 5. If the pin numbering is unclear, refer to the pictures above.

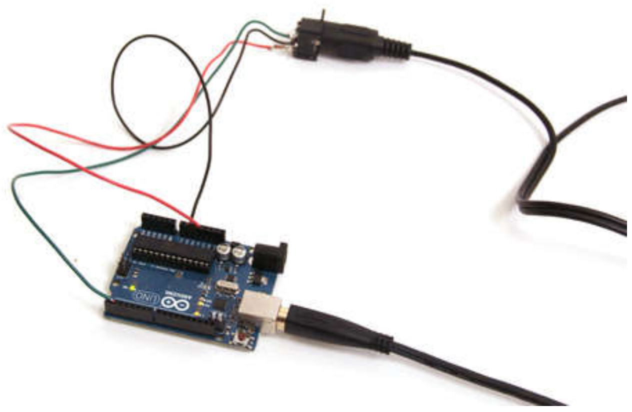






Step 4: Plug in MIDI Out

Connect the MIDI socket to a MIDI cable and plug the other end of the cable into your MIDI instrument of choice. I used a MIDI to USB cable and connected to my computer.



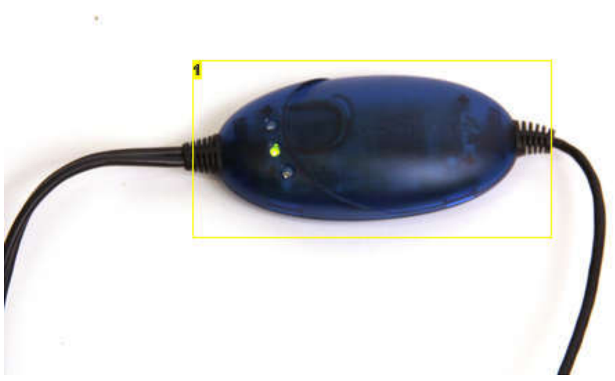


Image Notes

1. MIDI to usb

Step 5: Basic Note On, Note Off with Arduino

This code sends MIDI messages out arduino digital pin 1 using note on and note off commands.

As I explained in step 3, the MIDI commands for note on and note off are as follows:
noteON = 10010000 = 144
noteOFF = 10000000 = 128

Both of these commands are followed by two more bytes to make a complete MIDI message, the first is note and the second is velocity (for more info about what "note" and "velocity" mean check out my introductory MIDI instructable). Note and velocity can range from 0 to 127. In this example I used notes ranging from 50 to 69 (D3 to A4):
for (int note=50;note<70;note++){}
and I set the velocity to 100:
int velocity = 100;

So when the function MIDImessage() is called in the loop() of the arduino sketch, it sends the three bytes:
Serial.write(command);
Serial.write(MIDInote);
Serial.write(MIDIvelocity);
if the "command" in the MIDImessage() function is noteON then the note will start, if it is noteOFF the note will stop.

The code below plays the notes D3-A4 in a loop, it turns on a MIDI note for 300ms then wait 200ms before turning on the next note. I wrote a MaxMSP patch (you can download the runtime version for free) that displays the incoming MIDI messages and attached it below. Here is an example video:

```
/*
MIDI On/Off Messages
By Amanda Ghassaei
July 2012
http://www.instructables.com/id/Send-and-Receive-MIDI-with-Arduino/

* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
* (at your option) any later version.

*/

int vel = 100;//velocity of MIDI notes, must be between 0 and 127
//higher velocity usually makes MIDI instruments louder

int noteON = 144;//144 = 1001000 in binary, note on command
int noteOFF = 128;//128 = 10000000 in binary, note off command

void setup() {
 //  Set MIDI baud rate:
 Serial.begin(31250);
}

void loop() {
 for (int note=50;note<70;note++) {//from note 50 (D3) to note 69 (A4)
   MIDInoteON(note, vel);//turn note on
   delay(300);//hold note for 300ms
   MIDInoteOFF(note, vel);//turn note off
   delay(200);//wait 200ms until triggering next note
 }
}

//send MIDI message- turn note on
void MIDInoteON(int pitch, int velocity) {
 Serial.write(noteON);//send note on command
 Serial.write(pitch);//send pitch data
 Serial.write(velocity);//send velocity data
}
```

```
//send MIDI message- turn note off
void MIDInoteOFF(int pitch, int velocity) {
 Serial.write(noteOFF);//send note off command
 Serial.write(pitch);//send pitch data
 Serial.write(velocity);//send velocity data
}
```

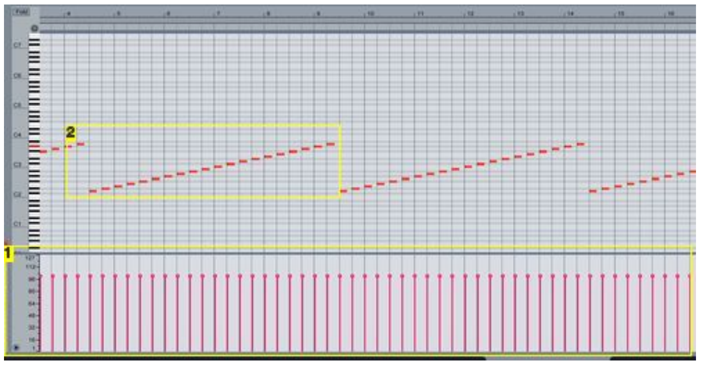
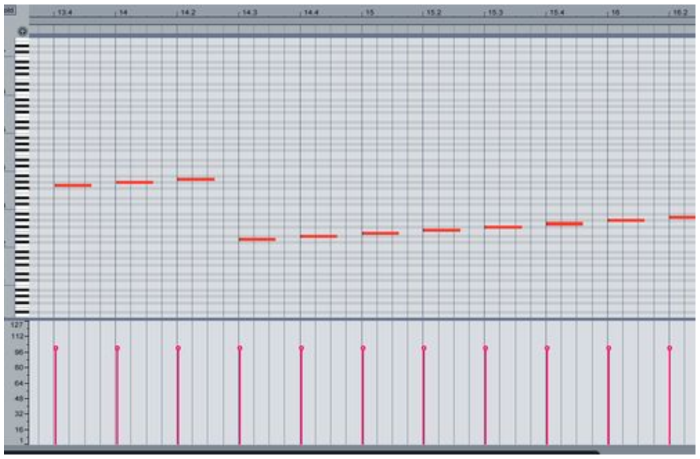
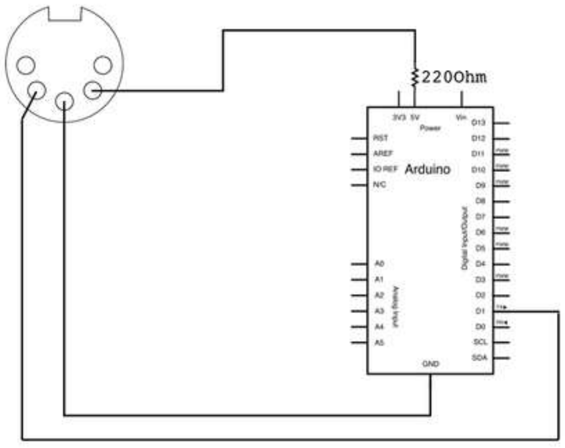


Image Notes
1. velocity set to constant 100
2. 20 notes, increasing by one semitone, with .3s duration





File Downloads

midiin.maxpat.zip (1 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'midiin.maxpat.zip']

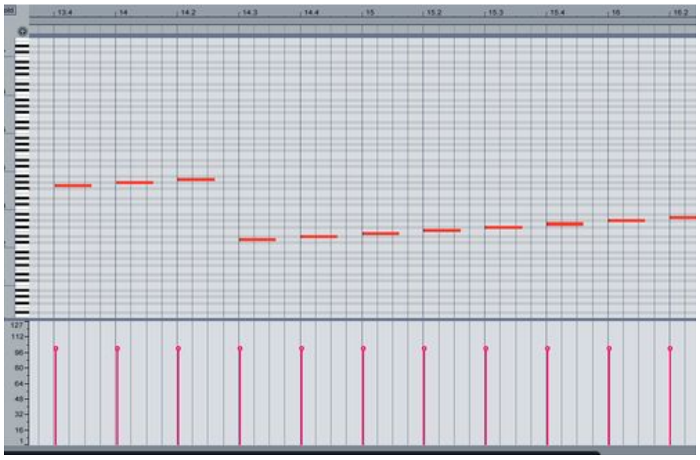
Step 6: Note Off with 0 Velocity

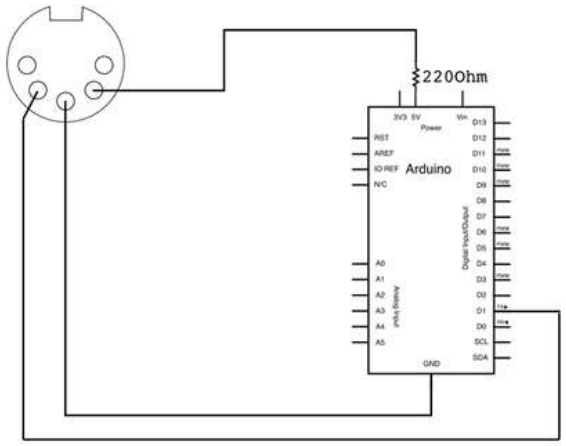
You may have noticed in the previous step's video, that the velocity appeared to set back to 0 when the note off message was sent. This is because a MIDI message with a note on command and velocity 0 is the same as a note off message. Sometimes when a software MIDI environment receives a note off message it will automatically translate it into a note on message with velocity 0 because they are the same. You will probably find that it is easier to program MIDI by sending these note on/velocity=0 messages rather than sending note off.

The code below does the exact same thing as the code in the last step, but it only uses note on commands. Essentially, I've replaced the following line:
MIDImessage(noteOFF, note, velocity);
with:
MIDImessage(noteON, note, 0);

```
/*
MIDI On Messages with 0 velocity
By Amanda Ghassaei
July 2012
http://www.instructables.com/id/Send-and-Receive-MIDI-with-Arduino/

* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
* (at your option) any later version.

*/

int velocity = 100;//velocity of MIDI notes, must be between 0 and 127
//(higher velocity usually makes MIDI instruments louder)
```

```
int noteON = 144;//144 = 1001000 in binary, note on command

void setup() {
 //  Set MIDI baud rate:
 Serial.begin(31250);
}

void loop() {
 for (int note=50;note<70;note++) {//from note 50 (D3) to note 69 (A4)
   MIDImessage(noteON, note, velocity);//turn note on
   delay(300);//hold note for 300ms
   MIDImessage(noteON, note, 0);//turn note off
   delay(200);//wait 200ms until triggering next note
 }
}

//send MIDI message
void MIDImessage(int command, int MIDInote, int MIDIvelocity) {
 Serial.write(command);//send note on or note off command
 Serial.write(MIDInote);//send pitch data
 Serial.write(MIDIvelocity);//send velocity data
}
```





Step 7: Variable Velocity and Arduino

In this code I've modified the variable velocity so that it increases with increasing note number. In the beginning of the loop() function the variable velocity is set to 20, then it is increased by five in each iteration of the for loop:
velocity += 5;
so for note = 50, velocity = 20, then for note = 51, velocity = 25, then for note = 52, velocity = 30... and so on.
once the end of the loop() function is reached, the velocity is reset back to 20.

Here's a video of the end result, notice how the volume increases with increasing velocity.

```
/*
MIDI note on messages with variable velocity
By Amanda Ghassaei
July 2012
http://www.instructables.com/id/Send-and-Receive-MIDI-with-Arduino/

* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
* (at your option) any later version.

*/

int noteON = 144;//144 = 1001000 in binary, note on command

void setup() {
 //  Set MIDI baud rate:
 Serial.begin(31250);
}

void loop() {
 int velocity = 20;//set velocity to 20
 for (int note=50;note<70;note++) {//from note 50 (D3) to note 69 (A4)
   MIDImessage(noteON, note, velocity);//turn note on
   delay(300);//hold note for 300ms
   MIDImessage(noteON, note, 0);//turn note off
   delay(200);//wait 200ms until triggering next note
   velocity += 5;//ad 5 to current velocity value
 }
}

//send MIDI message
void MIDImessage(int command, int MIDInote, int MIDIvelocity) {
 Serial.write(command);//send note on or note off command
 Serial.write(MIDInote);//send pitch data
 Serial.write(MIDIvelocity);//send velocity data
}
```

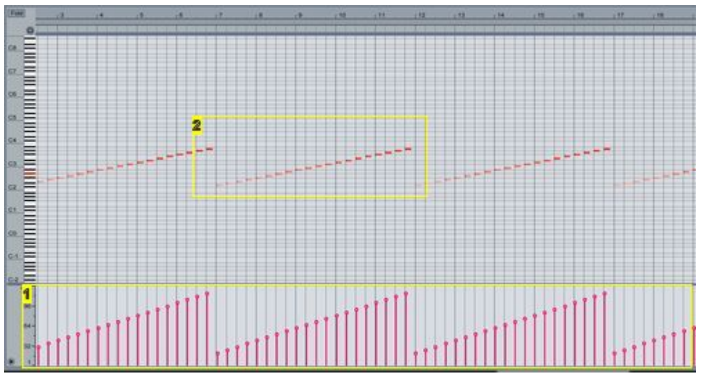
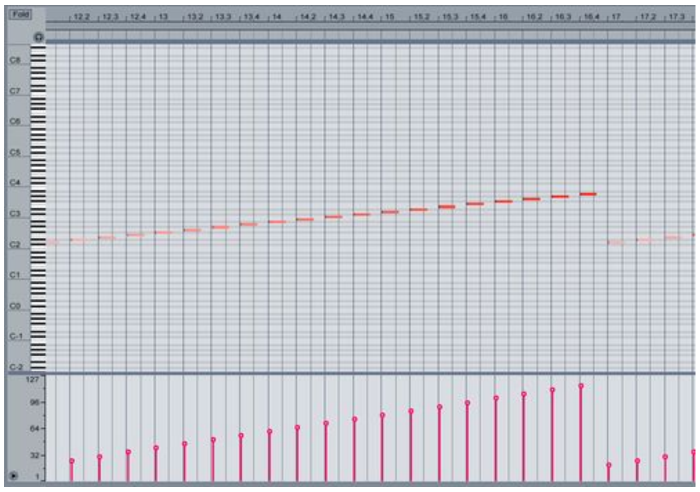
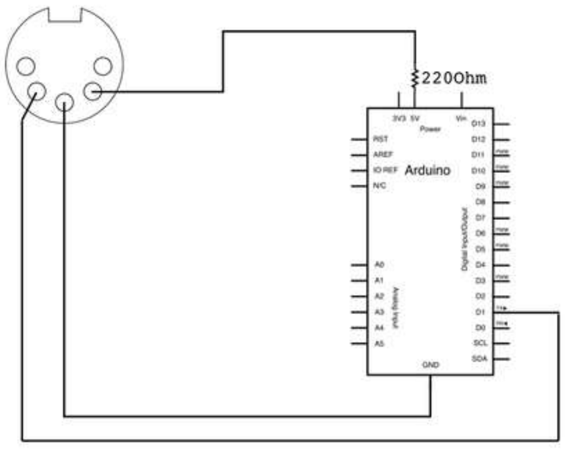


Image Notes
1. velocity increases by 5
2. 20 notes, increasing by one semitone, duration of 0.3s





Step 8: Pitchbend and Arduino

Now that you know how to control note on and note off, you can try pitchbend.

Pitchbend information is stored in 2 data bytes, and most significant byte (MSB) and a least significant byte (LSB). Each of these bytes contains only 7 bits of information. This means that all pitchbend information is stored in 14 bits, with the most significant 7 bits stored in the MSB and the least significant 7 bits stored in the LSB.

For most applications you will only find yourself changing pitchbend via the MSB and just setting the LSB to 0. In this case you have 7 bits of resolution for pitchbend (128 steps). In MIDI protocol pitchbend = 64 is no pitchbend, pitchbend greater than 64 is pitchbends the frequency up, and less than 64 pitchbends the frequency down.

In the example below a note played and held, then played again while the pitchbend increments from 64 to its max value of 127, then played a third time while the pitchbend increments from 64 to its min value of 0. This sequence is looped forever. The images above show the output in ableton from this Arduino sketch.

```
/*
MIDI Pitchbend (msb)
By Amanda Ghassaei
July 2012
http://www.instructables.com/id/Send-and-Receive-MIDI-with-Arduino/

* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
* (at your option) any later version.

*/



int noteON = 144;//144 = 1001000 in binary, note on command
//noteON data storage:
int note = 60;//middle c
int velocity = 100;//velocity of MIDI notes, must be between 0 and 127
//(higher velocity usually makes MIDI instruments louder)

int pitchbend = 224;//224 = 11100000 in binary, pitchbend command
//pitchbend data storage:
int lsb = 0;//least siginificant bit of pitchbend message
int msb = 0;//most significant bit of pitchbend message
```

```
void setup() {
 //  Set MIDI baud rate:
 Serial.begin(31250);
}

void loop() {
 //first play note w/o pitchbend
 MIDImessage(pitchbend, 0, 64);//reset pitchbend to 0 (zero pitchbend is lsb = 0, msb = 64)
 MIDImessage(noteON, note, velocity);//turn note on
 delay(700);//sustain note
 MIDImessage(noteON, note, 0);//turn note off
 delay(500);//wait 500ms until triggering next note

 //then play with pitchbend up
 MIDImessage(pitchbend, 0, 64);//reset pitchbend to 0 (zero pitchbend is lsb = 0, msb = 64)
 MIDImessage(noteON, note, velocity);//turn note on
 for (msb=64;msb<=127;msb++){//increase pitchbend  msb 64 (no pitchbend) to 127
   MIDImessage(pitchbend, lsb, msb);//send pitchbend message
   delay(10);
 }
 MIDImessage(noteON, note, 0);//turn note off
 delay(500);//wait 500ms until triggering next note

 //then play with pitchbend down
 MIDImessage(pitchbend, 0, 64);//reset pitchbend to 0 (zero pitchbend is lsb = 0, msb = 64)
 MIDImessage(noteON, note, velocity);//turn note on
 for (msb=64;msb>=0;msb--){//decrease pitchbend msb from 64 (no pitchbend) to 0;
   MIDImessage(pitchbend, lsb, msb);//send pitchbend message
   delay(10);
 }
 MIDImessage(noteON, note, 0);//turn note off
 delay(500);//wait 200ms until triggering next note
}

//send MIDI message
void MIDImessage(int command, int data1, int data2) {
 Serial.write(command);//send command byte
 Serial.write(data1);//send data byte #1
 Serial.write(data2);//send data byte #2
}
```

Below is a video demonstration of the code above. For this piece of code, pitchbend will be most noticeable in instruments with a long sustain, such as a string instrument, keep that in mind when testing the code for yourself.

You will most likely be fine using only 128 steps of pitchbend resolution, but in case you must use all 16384 steps, see the code below. Basically what I've done here is defined a variable called pitchbendVal, which varies from 0 to 16383. As I said below the "zero" pitchbend value is msb = 64 and lsb = 0. In binary this is:

MSB = 64 = 01000000
LSB = 0 = 0000000
(remember MSB and LSB are 7 bit numbers)

putting these values together we get:

1000000 0000000
MSB LSB

or

10000000000000
which translates to **8192** in decimal

so now the "zero" pitchbend value is 8192.

You'll also notice I had to break the variable pitchbendVal into two 7 bit parts to send out via MIDI message:
MIDImessage(pitchbend, (pitchbendVal&27), (pitchbendVal>>7));
the first part, pitchbendVal&127, returns the least significant 7 bits of pitchbendVal
the second part, pitchbendVal>>7, returns the most significant 7 bits of pitchbendVal
see & and >> on the Arduino reference page for more info.

```
/*
MIDI Pitchbend (full resolution)
By Amanda Ghassaei
July 2012
http://www.instructables.com/id/Send-and-Receive-MIDI-with-Arduino/

* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
* (at your option) any later version.

*/


int noteON = 144;//144 = 1001000 in binary, note on command
//noteON data storage:
int note = 60;//middle c
int velocity = 100;//velocity of MIDI notes, must be between 0 and 127
//(higher velocity usually makes MIDI instruments louder)
```

```
int pitchbend = 224;//224 = 11100000 in binary, pitchbend command
//pitchbend data storage:
int pitchbendVal = 8192;//value between 0 and 16383. zero pitchbend = 8192

void setup() {
 //  Set MIDI baud rate:
 Serial.begin(31250);
}

void loop() {
 //first play note w/o pitchbend
 pitchbendVal = 8192;//reset pitchbend to "0" (zero pitchbend is pitchbendVal = 8192)
 MIDImessage(pitchbend, (pitchbendVal&27), (pitchbendVal>>7));//send pitchbend message
 MIDImessage(noteON, note, velocity);//turn note on
 delay(700);//sustain note
 MIDImessage(noteON, note, 0);//turn note off
 delay(500);//wait 500ms until triggering next note

 //then play with pitchbend up
 pitchbendVal = 8192;//reset pitchbend to "0" (zero pitchbend is pitchbendVal = 8192)
 MIDImessage(pitchbend, (pitchbendVal&27), (pitchbendVal>>7));//send pitchbend message
 MIDImessage(noteON, note, velocity);//turn note on
 for (pitchbendVal=8192;pitchbendVal<16384;pitchbendVal++){//increase pitchbend from 8192 to 16383
   MIDImessage(pitchbend, (pitchbendVal&27), (pitchbendVal>>7));//send pitchbend message
   delay(1);
 }
 MIDImessage(noteON, note, 0);//turn note off
 delay(500);//wait 500ms until triggering next note

 //then play with pitchbend down
 pitchbendVal = 8192;//reset pitchbend to "0" (zero pitchbend is pitchbendVal = 8192)
 MIDImessage(pitchbend, (pitchbendVal&27), (pitchbendVal>>7));//send pitchbend message
 MIDImessage(noteON, note, velocity);//turn note on
 for (pitchbendVal=8192;pitchbendVal>=0;pitchbendVal--){//decrease pitchbend 8192 to 0;
   MIDImessage(pitchbend, (pitchbendVal&27), (pitchbendVal>>7));//send pitchbend message
   delay(1);
 }
 MIDImessage(noteON, note, 0);//turn note off
 delay(500);//wait 500ms until triggering next note
}

//send MIDI message
void MIDImessage(int command, int data1, int data2) {
 Serial.write(command);//send command byte
 Serial.write(data1);//send data byte #1
 Serial.write(data2);//send data byte #2
}
```

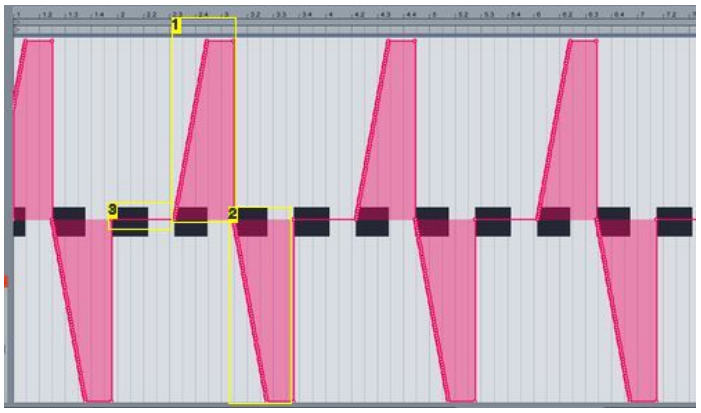


Image Notes
1. pitchbend increments up from 0 to max
2. pitchbend increments down from 0 to min
3. constant pitchbend of 64 = no pitchbend

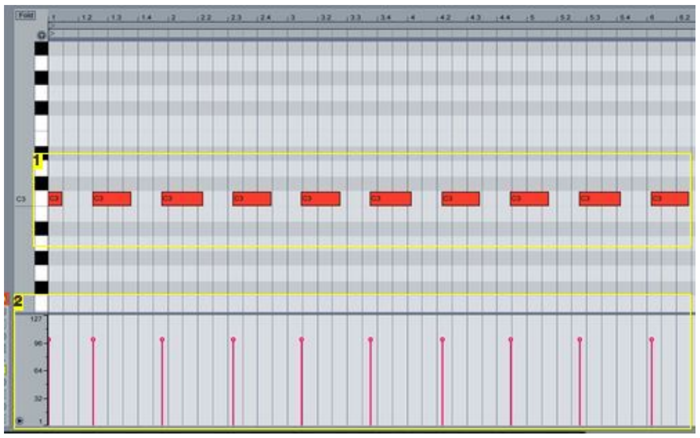
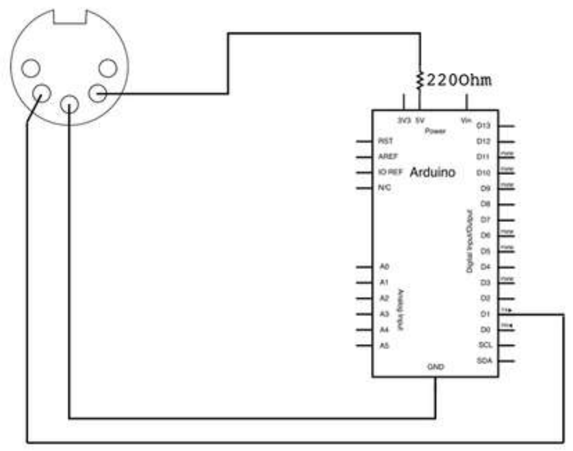


Image Notes
1. same note (middle c) repeated over and over
2. constant velocity



Step 9: Receive MIDI Messages with Arduino

Most Arduino MIDI projects send MIDI messages out, but you can also use the Arduino to receive MIDI data. Here are some ideas:

an Arduino synthesizer that uses MIDI messages to construct audio waveforms
a device which uses MIDI to trigger mechanical events, like the ringing of different sized bells
a MIDI to control voltage(CV) device- communication between MIDI and analog synthesizers

Parts List:
MIDI connector Digikey CP-2350-ND
220Ohm 1/4watt resistor Digikey CF14JT220RCT-ND
1N4148 diode Digikey1N4148-TAPCT-ND
10kOhm 1/4watt resistor Digikey CF14JT10K0CT-ND
470 Ohm 1/4watt resistor Digikey CF14JT470RCT-ND (I used 2x220 instead)
6N138 optocoupler Digikey 751-1263-5-ND

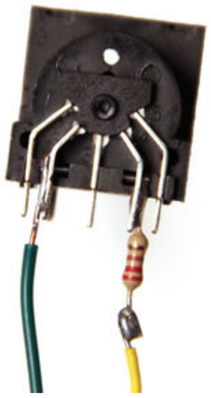
The hardware setup is slightly more complicated for receiving MIDI than it is for sending. As you can see in the schematic above, you have to set up an optoisolator in between the MIDI jack and the Arduino. If you are confused about the MIDI pin connections, refer to fig 1. I set this circuit up on a breadboard in figs 4 and 5.

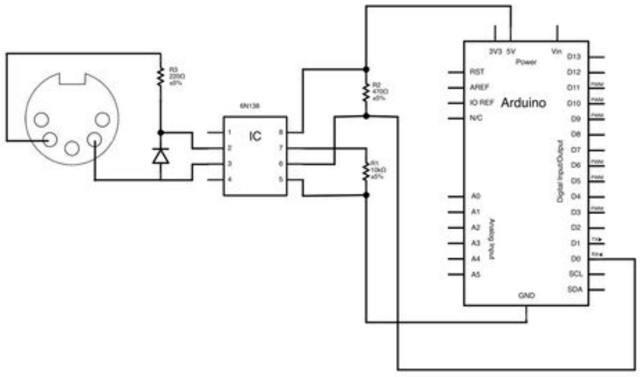
The following code receives these messages, reads them, and stores them appropriately. See the comments for more information.

```
/*Receive Midi
By Amanda Ghassaei
July 2012
http://www.instructables.com/id/Send-and-Receive-MIDI-with-Arduino/

* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
* (at your option) any later version.

*/

byte commandByte;
byte noteByte;
byte velocityByte;

void setup(){
 Serial.begin(31250);
}

void checkMIDI(){
 do{
   if (Serial.available()){
     commandByte = Serial.read();//read first byte
     noteByte = Serial.read();//read next byte
     velocityByte = Serial.read();//read final byte
   }
 }
 while (Serial.available() > 24);//when three bytes available
}


void loop(){
 checkMIDI();
}
```

To make sure that everything is working properly, try the code below. This code turns the led at pin 13 on briefly when it receives a note on message for MIDI note 60 (middle C). Notice how I included "& velocityByte>0" in the if statement- this makes sure that we are dealing with a note on statement, if you don't include this the light will blink for both note on and note on with velocity = 0 (note off) messages.

```
/*Receive MIDI and check if note = 60
By Amanda Ghassaei
July 2012
http://www.instructables.com/id/Send-and-Receive-MIDI-with-Arduino/

* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 3 of the License, or
* (at your option) any later version.

*/

byte commandByte;
byte noteByte;
byte velocityByte;

byte noteOn = 144;

//light up led at pin 13 when receiving noteON message with note = 60

void setup(){
 Serial.begin(31250);
 pinMode(13,OUTPUT);
 digitalWrite(13,LOW);
}

void checkMIDI(){
 do{
   if (Serial.available()){
     commandByte = Serial.read();//read first byte
     noteByte = Serial.read();//read next byte
     velocityByte = Serial.read();//read final byte
     if (commandByte == noteOn){//if note on message
       //check if note == 60 and velocity > 0
       if (noteByte == 60 & velocityByte > 0){
         digitalWrite(13,HIGH);//turn on led
       }
     }
   }
 }
 while (Serial.available() > 24);//when three bytes available
}


void loop(){
 checkMIDI();
 delay(100);
 digitalWrite(13,LOW);//turn led off
}
```





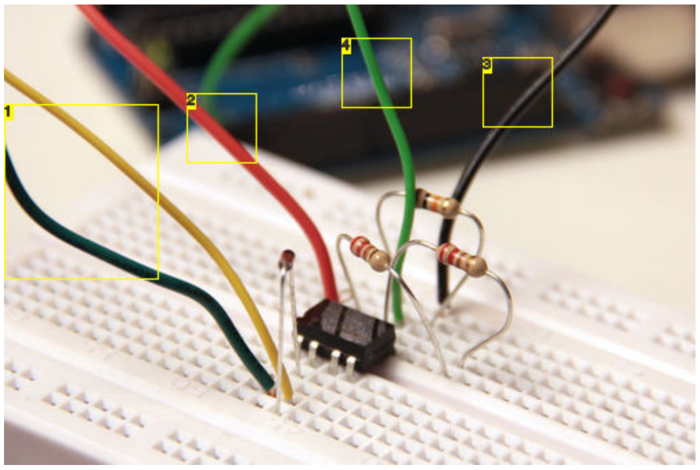


Image Notes
1. to midi jack
2. to Arduino 5V

3. to Arduino ground
4. to RX (digital pin 0)

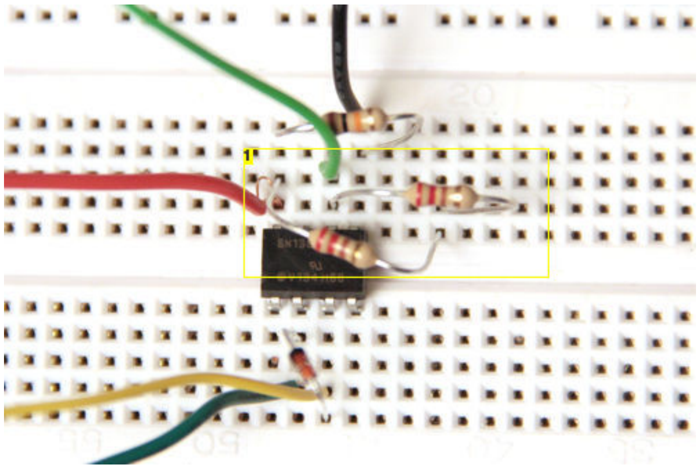


Image Notes
1. I used two 220ohm resistors instead of one 470ohm.

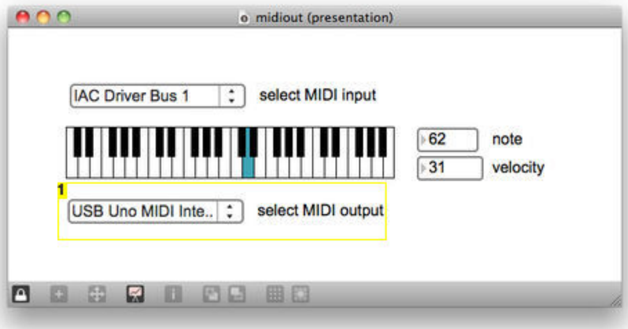


Image Notes
1. select Arduino here

File Downloads

midiout.maxpat.zip (1 KB)
[NOTE: When saving, if you see .tmp as the file ext, rename it to 'midiout.maxpat.zip']

Step 10: Examples

I recently built a MIDI controller with a built in accelerometer and gyroscope , as well as 16 backlit buttons. I'll be posting the full project soon (still need to finish enclosure and a few other things), but I've attached some example code that shows how I got the MIDI up and running. Here is a video of two programs I've written so far:

And here is the code for those applications:

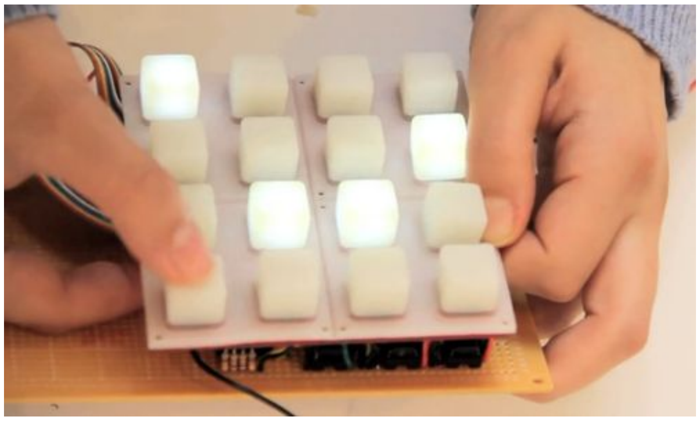
single pixel moving around, triggering MIDI (only uses x and y accelerometer):

```
//accelerometer test- single pixel
//by Amanda Ghassaei 2012
//http://www.instructables.com/id/Send-and-Receive-MIDI-with-Arduino/

/*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
*/


//pin connections
//#define ledLatchPin 6
//#define ledClockPin 5
//#define ledDataPin 7
//#define buttonLatchPin 4
//#define buttonClockPin 3
//#define buttonDataPin 2

//setup varibles for Gyroscope/Accelerometer
int xGyroRAW;
int yGyroRAW;
int xAccRAW;
int yAccRAW;
int zAccRAW;

byte xGyro;
byte yGyro;
byte xAcc;
byte yAcc;
byte zAcc;

//looping variables
byte i;
byte j;
byte k;

//storage for led states, 4 bytes
byte ledData[] = {0, 0, 0, 0};
//storage for buttons, 4 bytes
byte buttonCurrent[] = {0,0,0,0};
byte buttonLast[] = {0,0,0,0};
byte buttonEvent[] = {0,0,0,0};
byte buttonState[] = {0,0,0,0};
//button debounce counter- 16 bytes
byte buttonDebounceCounter[4][4];

//variables for accelerometer pixel movement
boolean firstPress = 1;
byte movingPixel[] = {0, 0, 0, 0};
byte yPosition;
byte xPosition;
int timeX = 0;
int timeY = 0;
boolean dirX = 0;
boolean dirY = 0;
byte lastX = 4;
byte lastY = 4;

//MIDI variables
int velocity = 100;
int noteON = 144;
int MIDIoffset = 60;
byte currentX;


void setup() {

 DDRD = 0xFA;//set pins D7-D4 as output, D2 as input

 Serial.begin(31250);//MIDI baud rate
//  Serial.begin(9600);

 cli();//stop interrupts

 //set timer1 interrupt at 1kHz
 TCCR1A = 0;// set entire TCCR1A register to 0
 TCCR1B = 0;// same for TCCR1B
 TCNT1  = 0;//initialize counter value to 0;
 // set timer count for 1khz increments
 OCR1A = 1999;// = (16*10^6) / (1000*8) - 1
 // turn on CTC mode
 TCCR1B |= (1 << WGM12);
 // Set CS11 bit for 8 prescaler
 TCCR1B |= (1 << CS11);
 // enable timer compare interrupt
 TIMSK1 |= (1 << OCIE1A);

 sei();//allow interrupts

}

ISR(TIMER1_COMPA_vect) {//Interrupt at freq of 1kHz
 timeX++;//increment timeX
 timeY++;//increment timeY
 shift();//send data to leds
}

// buttonCheck - checks the state of a given button.
//this buttoncheck function is largely copied from the monome 40h firmware by brian crabtree and joe lake
void buttonCheck(byte row, byte index)
{
 if (((buttonCurrent[row] ^ buttonLast[row]) &(1 << index)) &&   // if the current physical button state is different from the
 ((buttonCurrent[row] ^ buttonState[row]) &(1 << index))) {  // last physical button state AND the current debounced state

   if (buttonCurrent[row] &(1 << index)) {                      // if the current physical button state is depressed
     buttonEvent[row] = 1 << index;              // queue up a new button event immediately
     buttonState[row] |= (1 << index);                         // and set the debounced state to down.
 }
   else{
     buttonDebounceCounter[row][index] = 12;
   }  // otherwise the button was previously depressed and now
   // has been released so we set our debounce counter.
 }
 else if (((buttonCurrent[row] ^ buttonLast[row]) &(1 << index)) == 0 &&  // if the current physical button state is the same as
 (buttonCurrent[row] ^ buttonState[row]) &(1 << index)) {        // the last physical button state but the current physical
   // button state is different from the current debounce
   // state...
   if (buttonDebounceCounter[row][index] > 0 && --buttonDebounceCounter[row][index] == 0) {  // if the the debounce counter has
     // been decremented to 0 (meaning the
     // the button has been up for
     // kButtonUpDefaultDebounceCount
     // iterations///

     buttonEvent[row] = 1 << index;    // queue up a button state change event

     if (buttonCurrent[row] &(1 << index)){          // and toggle the buttons debounce state.
       buttonState[row] |= (1 << index);
     }
     else{
       buttonState[row] & ~(1 << index);
     }
   }
 }
}


void shift(){

 for (i=0;i<4;i++){

   buttonLast[i] = buttonCurrent[i];
   byte dataToSend = (1 << (i+4)) | (15 &~ledData[i]);

   // set latch pin low so the LEDs don't change while sending in bits
   PORTD&B10111111;//digitalWrite(ledLatchPin, LOW);
   // shift out the bits of dataToSend
   //shiftOut(ledDataPin, ledClockPin, LSBFIRST, dataToSend);
   for (j=0;j<8;j++){
     PORTD&B11011111;//digitalWrite(ledClockPin,LOW);
     //digitalWrite(ledDataPin,((dataToSend>>j)&));
     if ((dataToSend>>j)&1){
       PORTD|=B10000000;
     }
     else{
       PORTD&B01111111;
     }
     PORTD|=B00100000;//digitalWrite(ledClockPin,HIGH);
   }
   //set latch pin high so the LEDs will receive new data
   PORTD|=B01000000;//digitalWrite(ledLatchPin, HIGH);

   // SlowDown is put in here to waste a little time while we wait for the state of the output
   // pins to settle.  Without this time wasting loop, a single button press would show up as
   // two presses (the button and its neighbour)
   volatile int SlowDown = 0;

   while (SlowDown < 15)
   {
     SlowDown++;
   }

   //once one row has been set high, receive data from buttons
   //set latch pin high
   PORTD|=B00010000;//digitalWrite(buttonLatchPin, HIGH);
   //shift in data
   //buttonCurrent[i] = shiftIn(buttonDataPin, buttonClockPin, LSBFIRST) >> 3;
   for (j=0;j<4;j++){
     PORTD&B11110111;//digitalWrite(buttonClockPin,LOW);
     PORTD|=B00001000;//digitalWrite(buttonClockPin,HIGH);
   }
   for (j=0;j<4;j++){
     PORTD&B11110111;//digitalWrite(buttonClockPin,LOW);
     if ((PIND>>2)&1){//digitalRead(buttonDataPin)
       buttonCurrent[i]|=1<<j;
     }
     else{
       buttonCurrent[i]&~(1<<j);
     }
     PORTD|=B00001000;//digitalWrite(buttonClockPin,HIGH);
   }
   //latchpin low
   PORTD&B11101111;//digitalWrite(buttonLatchPin, LOW);

   for (k=0;k<4;k++){
     buttonCheck(i,k);
   }
 }

 //turn off leds- this way one row does not appear brighter than the rest

 // set latch pin low so the LEDs don't change while sending in bits
 PORTD&B10111111;//digitalWrite(ledLatchPin, LOW);
 // shift out 0
 //shiftOut(ledDataPin, ledClockPin, LSBFIRST, 0);
 for (j=0;j<8;j++){
   PORTD&B11011111;//digitalWrite(ledClockPin,LOW);
   PORTD&B01111111;
   PORTD|=B00100000;//digitalWrite(ledClockPin,HIGH);
 }
 //set latch pin high so the LEDs will receive new data
 PORTD|=B01000000;//digitalWrite(ledLatchPin, HIGH);
}

void checkFirstButton(){
 for (byte a=0;a<4;a++){
   if (buttonEvent[a]){
     for (byte b=0;b<4;b++){
       if (buttonState[a]&1<<b)){
         //toggle firstPress variable
         firstPress = 0;
         //display pressed pixel
         ledData[a] = buttonEvent[a];
         //store current position
         yPosition = a;
         xPosition = 1<<b;
         //reset timers
         timeX = 0;
         timeY = 0;
         return;
       }
     }
   }
 }
}

byte scaleAcc(int RAW){
 if (RAW<=10 & RAW>=-10){
   return 5;
 }
 else if (RAW<-10){
   if (RAW<-50){
     return 0;
   }
   else if (RAW<-40){
     return 1;
   }
   else if (RAW<-30){
     return 2;
   }
   else if (RAW<-20){
     return 3;
   }
   else{
     return 4;
   }
 }
 else if (RAW>10){
   if (RAW>50){
     return 10;
   }
   else if (RAW>40){
     return 9;
   }
   else if (RAW>30){
     return 8;
   }
   else if (RAW>20){
     return 7;
   }
   else{
     return 6;
   }
 }
}

void checkAccelerometer(){
 //read values
 xGyroRAW = analogRead(A1);
 yGyroRAW = analogRead(A0);
 xAccRAW = analogRead(A4);
 yAccRAW = analogRead(A3);
 zAccRAW = analogRead(A2);

 //offset data
 xGyroRAW = 317-xGyroRAW;
 yGyroRAW = 183-yGyroRAW;
 xAccRAW = 282-xAccRAW;
 yAccRAW = 282-yAccRAW;
 zAccRAW = 282-zAccRAW;

 if (xAccRAW>0){
   dirX = 1;
 }
 else{
   dirX = 0;
 }
 if (yAccRAW>0){
   dirY = 1;
 }
 else{
   dirY = 0;
 }

 //convert to 0-10
 xAcc = scaleAcc(xAccRAW);
 yAcc = scaleAcc(yAccRAW);
}

int getTime(byte acceleration){
 switch (acceleration){
   case 0://max - acceleration
   return 25;
   break;
   case 1:
   return 25;
   break;
   case 2:
   return 50;
   break;
   case 3:
   return 100;
   break;
   case 4:
   return 150;
   break;
   case 5://lying flat
   return 0;
   break;
   case 6:
   return 150;
   break;
   case 7:
   return 100;
   break;
   case 8:
   return 50;
   break;
   case 9:
   return 25;
   break;
   case 10://max + acceleration
   return 25;
   break;
 }
}

void moveXPixel(int timeComp){
 if (timeComp==0){
 }
 else{
   if (timeX>timeComp){
     timeX = 0;
     if (dirX){
       if (xPosition==8){
       }
       else{
         xPosition = xPosition<<1;
       }
     }
     else{
       if (xPosition==1){
       }
       else{
         xPosition = xPosition>>1;
       }
     }
   }
 }
}

void moveYPixel(int timeComp){
 if (timeComp==0){
 }
 else{
   if (timeY>timeComp){
     timeY = 0;
     if (dirY){
       if (yPosition==3){
       }
       else{
         yPosition = yPosition+=1;
       }
     }
     else{
       if (yPosition==0){
       }
       else{
         yPosition = yPosition-=1;
       }
     }
   }
 }
}

void checkMIDI(){
 //convert xPosition to decimal
 switch (xPosition){
   case 1:
   currentX = 0;
   break;
   case 2:
   currentX = 1;
   break;
   case 4:
   currentX = 2;
   break;
   case 8:
   currentX = 3;
   break;
 }
 //if pixel has moved send midi
 if (lastX != currentX || lastY != yPosition){
   MIDImessage(noteON,(lastX+5*lastY+MIDIoffset),0);//turn off last note
   MIDImessage(noteON,(currentX+5*yPosition+MIDIoffset),velocity);//turn on next note
 }
 lastX = currentX;
 lastY = yPosition;
}


void MIDImessage(int command, int MIDInote, int MIDIvelocity) {//send s a MIDI message
 Serial.write(command);//send note on or note off command
 Serial.write(MIDInote);//send pitch data
 Serial.write(MIDIvelocity);//send velocity data
}

void loop() {
 if (firstPress){
   checkFirstButton();
 }
 else{
   for (byte pixel=0;pixel<4;pixel++){
     if (pixel==yPosition){
       ledData[pixel]=xPosition;
     }
     else{
       ledData[pixel] = 0;
     }
   }
   checkAccelerometer();
   moveXPixel(getTime(xAcc));
   moveYPixel(getTime(yAcc));
   checkMIDI();
 }
}
```

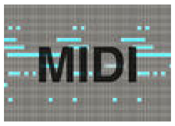
four pixels bouncing (only uses x accelerometer, uses x gyro to clear pixels):

```
//accelerometer test- bounce
//by Amanda Ghassaei 2012
//http://www.instructables.com/id/Send-and-Receive-MIDI-with-Arduino/

/*
* This program is free software; you can redistribute it and/or modify
* it under the terms of the GNU General Public License as published by
* the Free Software Foundation; either version 2 of the License, or
* (at your option) any later version.
*
*/

//pin connections
//#define ledLatchPin 6
//#define ledClockPin 5
//#define ledDataPin 7
//#define buttonLatchPin 4
//#define buttonClockPin 3
//#define buttonDataPin 2

//setup varibles for Gyroscope/Accelerometer
int xGyroRAW;
int yGyroRAW;
int xAccRAW;
int yAccRAW;
int zAccRAW;

byte xGyro;
byte yGyro;
byte xAcc;
byte yAcc;
byte zAcc;

//looping variables
byte i;
byte j;
byte k;

//storage for led states, 4 bytes
byte ledData[] = {0, 0, 0, 0};
//storage for buttons, 4 bytes
byte buttonCurrent[] = {0,0,0,0};
byte buttonLast[] = {0,0,0,0};
byte buttonEvent[] = {0,0,0,0};
byte buttonState[] = {0,0,0,0};
//button debounce counter- 16 bytes
byte buttonDebounceCounter[4][4];

//variables for accelerometer pixel movement
boolean firstPress[] = {0, 0, 0, 0};
byte movingPixel[] = {0, 0, 0, 0};
byte xPosition[4];
int timeX[] = {0, 0, 0, 0};
boolean dirX;
boolean dirY;
boolean prevDirX = 0;
boolean bounceDirection[]= {0, 0, 0, 0};
boolean toggle[] = {1, 1, 1, 1};
byte peakHeight[4];
byte lastX = 4;
byte lastY = 4;

//MIDI variables
int velocity = 100;
int noteON = 144;
int MIDIoffset = 60;
byte currentX;
byte note[] = {60, 64, 67, 72};


void setup() {

 DDRD = 0xFA;//set pins D7-D4 as output, D2 as input

 Serial.begin(31250);//MIDI baud rate

 cli();//stop interrupts

 //set timer1 interrupt at 1kHz
 TCCR1A = 0;// set entire TCCR1A register to 0
 TCCR1B = 0;// same for TCCR1B
 TCNT1  = 0;//initialize counter value to 0;
 // set timer count for 1khz increments
 OCR1A = 1999;// = (16*10^6) / (1000*8) - 1
 // turn on CTC mode
 TCCR1B |= (1 << WGM12);
 // Set CS11 bit for 8 prescaler
 TCCR1B |= (1 << CS11);
 // enable timer compare interrupt
 TIMSK1 |= (1 << OCIE1A);

 sei();//allow interrupts

}

ISR(TIMER1_COMPA_vect) {//Interrupt at freq of 1kHz
 for (byte a=0;a<4;a++){
   timeX[a]++;//increment each element of timeX
 }
 shift();
}

// buttonCheck - checks the state of a given button.
//this buttoncheck function is largely copied from the monome 40h firmware by brian crabtree and joe lake
void buttonCheck(byte row, byte index)
{
 if (((buttonCurrent[row] ^ buttonLast[row]) &(1 << index)) &&   // if the current physical button state is different from the
 ((buttonCurrent[row] ^ buttonState[row]) &(1 << index))) {  // last physical button state AND the current debounced state

   if (buttonCurrent[row] &(1 << index)) {                      // if the current physical button state is depressed
     buttonEvent[row] = 1 << index;              // queue up a new button event immediately
     buttonState[row] |= (1 << index);                         // and set the debounced state to down.
 }
   else{
     buttonDebounceCounter[row][index] = 12;
   }  // otherwise the button was previously depressed and now
   // has been released so we set our debounce counter.
 }
 else if (((buttonCurrent[row] ^ buttonLast[row]) &(1 << index)) == 0 &&  // if the current physical button state is the same as
 (buttonCurrent[row] ^ buttonState[row]) &(1 << index)) {        // the last physical button state but the current physical
   // button state is different from the current debounce
   // state...
   if (buttonDebounceCounter[row][index] > 0 && --buttonDebounceCounter[row][index] == 0) {  // if the the debounce counter has
     // been decremented to 0 (meaning the
     // the button has been up for
     // kButtonUpDefaultDebounceCount
     // iterations///

     buttonEvent[row] = 1 << index;    // queue up a button state change event

     if (buttonCurrent[row] &(1 << index)){          // and toggle the buttons debounce state.
       buttonState[row] |= (1 << index);
     }
     else{
       buttonState[row] & ~(1 << index);
     }
   }
 }
}


void shift(){

 for (i=0;i<4;i++){

   buttonLast[i] = buttonCurrent[i];

   byte dataToSend = (1 << (i+4)) | (15 &~ledData[i]);

   // set latch pin low so the LEDs don't change while sending in bits
   PORTD&B10111111;//digitalWrite(ledLatchPin, LOW);
   // shift out the bits of dataToSend
   //shiftOut(ledDataPin, ledClockPin, LSBFIRST, dataToSend);
   for (j=0;j<8;j++){
     PORTD&B11011111;//digitalWrite(ledClockPin,LOW);
     //digitalWrite(ledDataPin,((dataToSend>>j)&));
     if ((dataToSend>>j)&1){
       PORTD|=B10000000;
     }
     else{
       PORTD&B01111111;
     }
     PORTD|=B00100000;//digitalWrite(ledClockPin,HIGH);
   }
   //set latch pin high so the LEDs will receive new data
   PORTD|=B01000000;//digitalWrite(ledLatchPin, HIGH);

   // SlowDown is put in here to waste a little time while we wait for the state of the output
   // pins to settle.  Without this time wasting loop, a single button press would show up as
   // two presses (the button and its neighbour)
   volatile int SlowDown = 0;

   while (SlowDown < 15)
   {
     SlowDown++;
   }

   //once one row has been set high, receive data from buttons
   //set latch pin high
   PORTD|=B00010000;//digitalWrite(buttonLatchPin, HIGH);
   //shift in data
   //buttonCurrent[i] = shiftIn(buttonDataPin, buttonClockPin, LSBFIRST) >> 3;
   for (j=0;j<4;j++){
     PORTD&B11110111;//digitalWrite(buttonClockPin,LOW);
```

```
    PORTD|=B00001000;//digitalWrite(buttonClockPin,HIGH);
  }
  for (j=0;j<4;j++){
    PORTD&B11110111;//digitalWrite(buttonClockPin,LOW);
    if ((PIND>>2)&1){//digitalRead(buttonDataPin)
      buttonCurrent[i]|=1<<j;
    }
    else{
      buttonCurrent[i]&~(1<<j);
    }
    PORTD|=B00001000;//digitalWrite(buttonClockPin,HIGH);
  }
  //latchpin low
  PORTD&B11101111;//digitalWrite(buttonLatchPin, LOW);

  for (k=0;k<4;k++){
    buttonCheck(i,k);
  }
 }

 //turn off leds- this way one row does not appear brighter than the rest

 // set latch pin low so the LEDs don't change while sending in bits
 PORTD&B10111111;//digitalWrite(ledLatchPin, LOW);
 // shift out 0
 //shiftOut(ledDataPin, ledClockPin, LSBFIRST, 0);
 for (j=0;j<8;j++){
   PORTD&B11011111;//digitalWrite(ledClockPin,LOW);
   PORTD&B01111111;
   PORTD|=B00100000;//digitalWrite(ledClockPin,HIGH);
 }
 //set latch pin high so the LEDs will receive new data
 PORTD|=B01000000;//digitalWrite(ledLatchPin, HIGH);
}

void checkPress(byte Y){
 if (buttonEvent[Y]){
   for (byte b=0;b<4;b++){
     if (buttonState[Y]&1<<b)){
       //toggle firstPress variable
       firstPress[Y] = 1;
       //display pressed pixel
       ledData[Y] = (1<<b);
       //store current position
       xPosition[Y] = (1<<b);
       //store peak height
       peakHeight[Y] = (1<<b);
       //reset timers
       timeX[Y] = 0;
       return;
     }
   }
 }
}

byte scaleAcc(int RAW){
 if (RAW<=10 & RAW>=-10){
   return 5;
 }
 else if (RAW<-10){
   if (RAW<-50){
     return 0;
   }
   else if (RAW<-40){
     return 1;
   }
   else if (RAW<-30){
     return 2;
   }
   else if (RAW<-20){
     return 3;
   }
   else{
     return 4;
   }
 }
 else if (RAW>10){
   if (RAW>50){
     return 10;
   }
   else if (RAW>40){
     return 9;
   }
   else if (RAW>30){
     return 8;
   }
   else if (RAW>20){
     return 7;
   }
   else{
     return 6;
   }
 }
}

void checkAccelerometerGyro(){
 //read values
 xGyroRAW = analogRead(A1);
 yGyroRAW = analogRead(A0);
 xAccRAW = analogRead(A4);
 yAccRAW = analogRead(A3);
 zAccRAW = analogRead(A2);

 //offset data
 xGyroRAW = 317-xGyroRAW;
 yGyroRAW = 183-yGyroRAW;
 xAccRAW = 282-xAccRAW;
 yAccRAW = 282-yAccRAW;
 zAccRAW = 282-zAccRAW;

 //convert to 0-10
 xAcc = scaleAcc(xAccRAW);
 yAcc = scaleAcc(yAccRAW);

 if (xAccRAW>5){
   dirX = 1;
 }
 else if (xAccRAW<5){
   dirX = 0;
 }
 if (yAccRAW>5){
   dirY = 1;
 }
 else if (yAccRAW>5){
   dirY = 0;
 }

}

int getTime(byte acceleration){
 switch (acceleration){
   case 0://max - acceleration
   return 100;
   break;
   case 1:
   return 100;
   break;
   case 2:
   return 150;
   break;
   case 3:
   return 200;
   break;
   case 4:
   return 250;
   break;
   case 5://lying flat
   return 0;
   break;
   case 6:
   return 250;
   break;
   case 7:
   return 200;
   break;
   case 8:
   return 150;
   break;
   case 9:
   return 100;
   break;
   case 10://max + acceleration
   return 100;
   break;
 }
}

void moveXPixel(byte Y, int timeComp){
 if (timeComp==0){
 }
 else{
   if (timeX[Y]>timeComp){
     timeX[Y] = 0;
     if (dirX){
       if (peakHeight[Y]==8&&xPosition[Y]==8){
         if(toggle[Y]){
           MIDImessage(noteON,note[Y],0);//send midi
           toggle[Y]=0;
           ledData[Y]=0;
         }
         else{
           MIDImessage(noteON,note[Y],velocity);
           toggle[Y]=1;
         }
       }
       else{
         toggle[Y]=1;
         if (xPosition[Y]==peakHeight[Y]){//if at peak
           bounceDirection[Y]=1;//falling
           MIDImessage(noteON,note[Y],0);//turn note off
         }
         if (xPosition[Y]==8){//if hitting bottom
           bounceDirection[Y]=0;//rising
           MIDImessage(noteON,note[Y],velocity);//turn note on
         }
         if (xPosition[Y]==1){
           bounceDirection[Y]=1;
         }
         if (bounceDirection[Y]){
           xPosition[Y] = xPosition[Y]<<1;
         }
         else{
           xPosition[Y] = xPosition[Y]>>1;
         }
       }
     }
     else{
       if (peakHeight[Y]==1&&xPosition[Y]==1){
         if(toggle[Y]){
           MIDImessage(noteON,note[Y],0);//send midi
           toggle[Y]=0;
           ledData[Y]=0;
         }
         else{
           MIDImessage(noteON,note[Y],velocity);
           toggle[Y]=1;
         }
       }
       else{
         toggle[Y]=1;
         if (xPosition[Y]==peakHeight[Y]){//if at peak
           bounceDirection[Y]=0;//falling
           MIDImessage(noteON,note[Y],0);//turn note off
         }
         if (xPosition[Y]==8){
           bounceDirection[Y]=0;
         }
         if (xPosition[Y]==1){//if hitting bottom
           bounceDirection[Y]=1;//rising
           MIDImessage(noteON,note[Y],velocity);//turn note on
         }
         if (bounceDirection[Y]){
           xPosition[Y] = xPosition[Y]<<1;
         }
         else{
           xPosition[Y] = xPosition[Y]>>1;
         }
       }
     }
   }
 }
}

void shake2Clear(){
 if (abs(xGyroRAW)>300){
   for (byte a=0;a<4;a++){
     firstPress[a]=0;
     ledData[a]=0;
   }
 }
}

void MIDImessage(int command, int MIDInote, int MIDIvelocity) {//send s a MIDI message
 Serial.write(command);//send note on or note off command
 Serial.write(MIDInote);//send pitch data
 Serial.write(MIDIvelocity);//send velocity data
}

void loop() {
 checkAccelerometerGyro();
 shake2Clear();
 for (byte column=0;column<4;column++){
   checkPress(column);
   if (firstPress[column]){
   moveXPixel(column, getTime(xAcc));
   if (toggle[column]){
     ledData[column]= xPosition[column];
   }
   }
 }
}
```

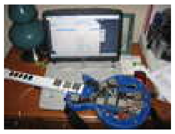


Related Instructables



What is MIDI? by amandaghassaei

Converting a rescued toy into a MIDI controller by zen.webb



Portable MIDI Tone Generator On-the-Cheap by lewisb42

Adding MIDI to Old Home Organs by uhclem

Electronic Instrument by amandaghassaei

Arcade Button MIDI Controller by fraganator