Accelerometer based Wireless Intuitive MIDI Controller

Manjishta Kainth[1], Sanket Kulkarni[2], Forum Mehta[3]

[1]Undergraduate student, Electronics and Telecommunication Department,
DJSCE, Vile Parle (West), Mumbai 400056
manjishta.k@gmail.com

[2]Undergraduate student, Electronics and Telecommunication Department,
DJSCE, Vile Parle (West), Mumbai 400056
sanket26k@gmail.com

[3]Undergraduate student, Electronics and Telecommunication Department,
DJSCE, Vile Parle (West), Mumbai 400056
forummehta101@gmail.com

Abstract: The growing research and development in Music & Technology has put great emphasis on creative development and deployment of innovative musical technologies which adds a completely new dimension to how we experience music. In recent years, MIDI Controllers have been used extensively to control various parameters of a live performance or a studio production. The MIDI Controllers available in market are generally employed with a number of sliders, knobs, buttons and may or may not include a piano-keyboard. In this paper, we propose a MIDI controller based on an accelerometer to enhance intuitive motion based non-verbal experience which can be used with or without company of the traditional button-slider MIDI controller. We implement this Controller using a 3-D accelerometer module ADXL 330 & Arduino AT Mega 8 as a Human Interface Device to add the much needed portability for system enhancement.

Keywords: Accelerometer, Human Interface Device, MIDI Controller, Music Technology.

1. Introduction

With digital advancements, there is a voracious need to revolutionize the music industry as well as interface instruments with technology. For easy utility by music enthusiasts such as remote controlling, a wireless project such as this satisfies the urge to understand music note generation from a technical point of view using MIDI notes.

MIDI is a digital language shared between electronic instruments and the computer software to lead to ample music possibilities. MIDI stands for Musical Instrument Digital Interface and is specifically used to generate music in real-time depending on the user's need; and imagination. A MIDI note is equivalent to a musical action such as strumming a guitar or pressing a key on a piano. The notes connect the software to a library of musical functions used by MIDI putting an entire orchestra at our disposal. Safe to say, MIDI is a standard useful in controlling any (and all) music devices available in the market. MIDI notes can be generated on the software or on an instrument termed as a MIDI Controller and played on the sound card of the user's personal computer.

2. Literature Review

2.1 Background

MIDI (Music Instrument Digital Interface) is an 8-bit binary serial protocol that operates at 31,250 bits per second. MIDI bytes are divided into two types: command bytes and data bytes**.** Command bytes are always 128 or greater, or 0x80 to 0xFF in hexadecimal. Data bytes are always less than 127, or 0x00 to 0x7F in hex. Commands include extra functions such as note on, note off, pitch bend on or off, and so forth. Data bytes include things like the pitch of the note to play, the velocity, (loudness of the note), amount of pitch bend and so forth. MIDI data is usually notated in hexadecimal because MIDI banks and instruments are grouped in groups of 16.

2.2 Message Structure

MIDI Messages comprise a STATUS byte followed by DATA bytes. Messages are roughly divided into two main categories: Channel and System. Channel messages contain a four bit channel number encoded into the Status byte which addresses the message specifically to one of the sixteen channels. System messages are not encoded with channel numbers but are further divided into three main types: System Common, System Real Time and System Exclusive. System Real Time messages are all single bytes and may be interleaved between any other messages. Any Status byte except System Real Time may terminate a System Exclusive Block.

For channel messages only, the Status byte may be omitted if it would otherwise repeat the last Status byte sent. This is commonly used to reduce the length of Note On/Note Off sequences or Continuous Controller Movements, like for sending accelerometer data continuously until a MIDI off is transmitted.

2.3 Arduino

Arduino has been the brain for various projects for over the past few years – all thanks to the user friendly features like inexpensiveness, cross-platform compatibility, lucid programming environment**,** open-source and extensible software & hardware.

2.4 The Arduino MIDI library

ISSN 2229-5518

This Arduino library allows I/O communications on the Arduino serial ports. We can send and receive messages of several kinds (such as System Exclusive, Real-time etc.) It will help us interface Arduino with other MIDI devices.

MIDI.begin(MIDI_CHANNEL_OMNI): This initializes the Midi Library. This parameter sets the library to listen to all Midi Channels.
MIDI.begin(n) would set it to listen to Channel n only.
Here, n starts from channel 0 to channel 15.
MIDI.setHandleNoteOn(MyHandleNoteOn);
This acts like an import command. The Arduino Midi Library uses 'Callbacks' i.e. when a Midi event occurs, the Library will call a function to handle it. This command tells the Library to call the 'MyHandleNoteOn' function when a 'Note On' Midi event is detected. There are many callback functions in the Library to handle the many types of Midi events (Clock, Pitch Bend, Program Change, etc.). We use the MIDI.set command to point to the functions we require.
Void MIDImessage(byte channel, byte pitch, byte velocity). This is the function we created to be called when a Midi Note On event is detected. This is the 'essence' of our program.
MIDI.Read() is the only function in the main loop of the program. It just checks the input buffer for any received Midi commands and passes them to the correct function.

2.5 Accelerometer

Accelerometer measure acceleration. It is extensively used to easily calculate the inclination of an object with respect to the ground. The accelerometer by itself uses very little current, so it can be plugged into our Arduino board and run directly off of the output from the digital output pins. To do this, we've used three of the analog input pins as digital I/O pins, for power and ground to the accelerometer, and for the self-test pin. We'll use the other three analog inputs to read the accelerometer's analog outputs.

ADXL 3xx Series sensors enables us to gather real time physical control over musical parameters such as volume, pitch, timbre.

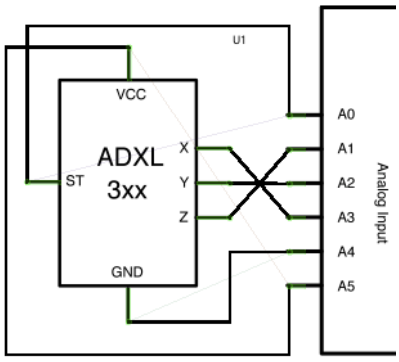

Figure 1: Accelerometer pin-diagram

3. Implementation

The basic functioning of the prototype is as follows:

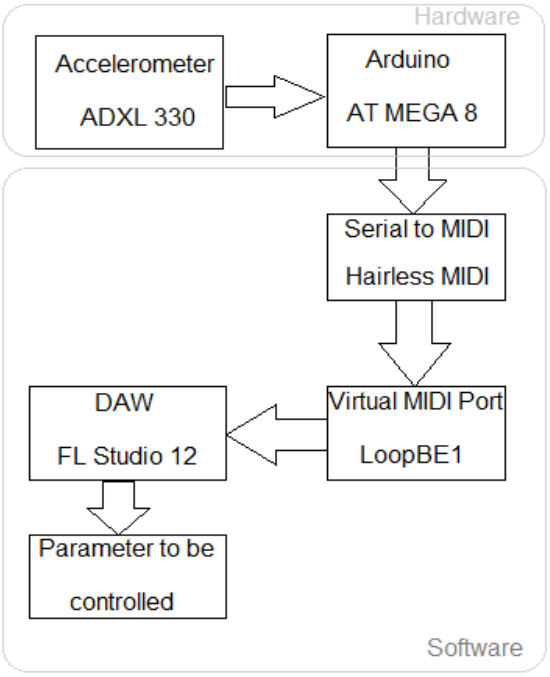

Figure 2: Basic Block Diagram

3.1 Interface accelerometer with Arduino

The physical connections of the accelerometer are done as follows:

ADXL Ground- Arduino Ground

ADXL Power - Arduino V(ref) 3.3V

ADXL Xout- Analog Input 1

ADXL Yout- Analog Input 2

ADXL Zout - Analog Input 3

This can be achieved by defining each pin–

const int ap1 = A1;

const int ap2 = A2;

const int ap3 = A3;

const int groundpin = A4;

const int powerpin = A5;

& then assigning required pins as ground and power appropriately.

pinMode(groundpin, OUTPUT);

pinMode(powerpin, OUTPUT);

digitalWrite(groundpin, LOW);

digitalWrite(powerpin, HIGH);

Now, to setup the Vref after connecting 3.3V to AREF we add

analogReference(EXTERNAL);

Thus, we have setup the accelerometer to be functional.

3.2 Analog input

The Arduino board contains a 6 channel 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. This yields a resolution between readings of: 5 volts / 1024 units or, .0049 volts (4.9 mV) per unit. The input range and resolution is

changed using analogReference(). Thus, 3.3V maps to 1023 if we have 3.3V Aref.

It takes about 100 microseconds (0.0001 s) to read an analog input, so the maximum reading rate is about 10,000 times a second.

The syntax for reading analog voltage at a pin is

analogRead(pin).

If the analog input pin is not connected to anything, the value returned by analogRead() will fluctuate based on a number of factors like the values of the other analog inputs or how close is the board to other electro-magnetic changes.

sv1 = analogRead(ap1);

sv2 = analogRead(ap2);

sv3 = analogRead(ap3);

Here, we store the output integer between 0 and 1023 in 3 variables.

3.3 Mapping range for MIDI

Mapping function re-maps a number from one range to another. That is, a value of from Low range 1 would get mapped to Low Range 2, a value of from High Range 1 to High Range 2, values in-between to values in-between, etc.

Does not constrain values to within the range, because out-of-range values are sometimes intended and useful. The constrain() function is used either before or after this function, if limits to the ranges are desired. The syntax for mapping range is:

map(value, Low Range 1, High Range 1, Low Range 1, High Range 2).

The map() function uses integer math so will not generate fractions, when the math might indicate that it should do so. Fractional remainders are truncated, and are not rounded or averaged. We map the range from 0 to 1023 to 0 to 127, because MIDI data ranges valid only from 0 to 127.

ov(n) = map(sv(n), 0, 1023, 0, 127);

n=1,2,3

Since we can only send data bits now, we manually specify note ON and note OFF messages.

byte noteON = 144;

We use a note with velocity 0 as a note OFF command.

3.4 MIDI message function

We define a function to actually send each message. It is called every time MIDI notes need to be transmitted. The function consists of 3 parameters namely command, and 2 data bytes. The function looks like:

```
void MIDImessage(byte command, byte data1, byte data2)
{
  Serial.write(command);
  Serial.write(data1);
  Serial.write(data2);
}
```

Here, the command used can be noteON we defined earlier. Data Byte 1 is used to convey the pitch of the MIDI note transferred. Thus, for data1 = 0, C0 is transmitted. Data Byte 2 conveys the velocity of the MIDI note. Since velocity cannot be greater than 100, all the values above 100 are assumed to be 100. Velocity is the loudness of the note currently being played. It can also be used to control other parameters as filter cut-off's or pre-fader gain.

We can use 2 accelerometers simultaneous as pitch and velocity controllers while calling this function as follows.

MIDImessage(noteON, ov1, ov2);

3.5 Serial to MIDI

The Serial to MIDI converter is a software solution to get our computer's serial port (or virtual serial port over USB) talking with the MIDI software and hardware.

Normally, to use an Arduino or other micro-controller with our MIDI software we had to build a MIDI-in and MIDI-out circuit with a few parts and an opto-coupler. Easy enough, but then we would typically need a MIDI to USB adaptor to connect it to the computer. With the software solution, this is possible with micro-controller like Arduino & a USB connector. Hairless MIDI Serial Bridge is the easiest way to connect serial devices (like Arduinos) to send and receive MIDI signals. It is open-source software for Mac OS X, Windows & Linux. Once Serial to MIDI is configured, it will

a. Take "MIDI" incoming serial data and forward it to the desired MIDI port.

b. Take MIDI data coming from the chosen MIDI port and forward it out of the serial port.

Since, an actual MIDI port needs a hardware MIDI connection, we need to create a virtual MIDI port.

3.6 Virtual MIDI Port

A virtual MIDI port creates a fake freely nameable MIDI port. It sends the data received from the Serial to MIDI converter to the DAW.

Virtual MIDI port is an internal MIDI device for transferring MIDI data between computer programs. Basically it acts an "invisible cable" to connect a MIDI out port of an application to any other application´s MIDI in port.

All MIDI data sent to the program´s output is channeled to the receiving applications in real time.

We may connect up to 8 applications to LoopBe in port and up to 8 applications to the out port, all sending and receiving at the same time.

3.7 Digital Audio Workstation

A digital audio workstation (D.A.W.) is an electronic device or computer software application for recording, editing and producing audio files.

"DAW" can simply refer to the software itself, but traditionally, a computer-based DAW has four basic components: a computer, either a sound card or audio interface, digital audio editor software, and at least one input device for adding or modifying data. The computer acts as a host for the sound card/audio interface, while the software provides the interface and functionality for audio editing.

Computer-based DAWs have extensive recording, editing, and playback capabilities (some even have video-related features).

As software systems, DAWs could be designed with any user interface, but generally they are based on a multitrack tape recorder metaphor, making it easier for recording engineers and musicians already familiar with using tape recorders to become familiar with the new systems. Therefore, computer-based DAWs tend to have a standard layout that includes transport controls (play, rewind, record, etc.), track controls and a mixer, and a waveform display.

FL Studio supports up to 100 MIDI inputs and outputs. Thus, one input port can be configured to the output of the MIDI notes from the Arduino through the virtual MIDI port.

Thus, the MIDI data can be used to control pitch and/or velocity of any parameters inside the DAW.

4. Conclusion

An approach is presented to study how motions produce sounds in real-time based on MIDI notes and a wireless link. The results are optimized solely for ease of accessibility. The device is based on the concept of human machine interaction technique as it utilizes the accelerometer values i.e., hand gestures to define different effects applied to music as per requirements extending the use of the device from a simple studio production to a live-performance stage concert. The advantage of the project lies in the versatility of operations as the user can map effects with the accelerometer values depending on the requirement. Gestures work as a form of non-verbal communication to send instructions to D.A.W. FL Studio as studied previously. Moreover Arduino libraries, MIDI messages and functions were studied that aid in the research.

The project holds a massive future scope in the music industry as well as advancement in motion technology for its various advantages such as portability, low cost and easy use for having an edge over the primarily existing MIDI Controllers. The project thrives on the convenience of hand gestures transmitted over wireless connection replacing the hassle of knobs and sliders.

References

[1] Arduino MIDI library v4.2, 2014
www.github.com/FortySevenEffects/arduino_midi_library

[2] Zohra Aziz Ali Manjiyani, Renju Thomas Jacob, Keerthan Kumar R, Babu Varghese, School of Electronics, VIT "Development of MEMS Based 3-Axis Accelerometer for Hand Movement Monitoring", February 2014.

[3] FL Studio, Image Line Software 1998-2015
www.image-line.com/flstudio

[4] Steven Campbell, James Cook University "An Ultrasonic Gestural MIDI Controller", Proceedings of Australian Computer Music Conference, 2005.

Author Profile

Manjishta Kainth is currently an undergraduate student at Dwarkadas J. Sanghvi College of Engineering, Mumbai. She belongs to the Electronics and Telecommunication Department.

Sanket Kulkarni is currently an undergraduate student at Dwarkadas J. Sanghvi College of Engineering, Mumbai. He belongs to the Electronics and Telecommunication Department.

Forum Mehta is currently an undergraduate student at Dwarkadas J. Sanghvi College of Engineering, Mumbai. She belongs to the Electronics and Telecommunication Department.