Lab 13 Controllers and Modulation

Korg nanoPAD2 MIDI Controller

Connect the USB cable to the nanoPAD2 MIDI Controller and then to the computer. Make sure the all top row mini buttons are off, except for SCENE that should light up number one.

Setup

If you haven't already done so, download and unzip m208Lab13.zip to your Desktop.

Mac Only - Open Au Lab

In order to use MIDI output on the Mac you must open Au Lab before opening miniAudicle. This is not necessary if you are using Windows.

Open miniAudicle

Quit and restart miniAudicle. Restarting miniAudicle may not always be necessary, but if your program stops responding to MIDI input that's the first thing to try. Set m208Lab13 to the working directory in miniAudicle. If you've been working with other MIDI devices, remove all shreds and save any files you've been working on.

Identify the MIDI Input and MIDI Output Device Numbers

Open Terminal and run the chuck --probe command to list the available Audio and MIDI input and output devices. This command works on both Mac and Windows but must be run from a Terminal window. The MIDI devices are

shown at the end of the list and the device number is shown in brackets. There is a second option but its Mac only: choose Device Browser from the miniAudicle Window menu.

If you've already done the QX25 lab you can reuse some of the following code.

ChucK MIDI Event Documentation

ChucK contains built-in MIDI classes to allow for interaction with MIDI based software or devices.

```
MidiIn min;
MidiMsg msg;

// open midi receiver, exit on fail
if ( !min.open(0) ) me.exit();

while( true )
{
    // wait on midi event
    min => now;

    // receive midimsg(s)
    while( min.recv( msg ) )
    {
        // print content
     <<< msg.data1, msg.data2, msg.data3 >>>;
    }
}
...
```

MidiIn is a subclass of **Event**, and as such can be ChucKed to **now**. MidiIn then takes a MidiMsg object to its **.recv()** method to access the MIDI data.

As a default, MidiIn events trigger the **broadcast()** event behavior.

http://chuck.cs.princeton.edu/doc/language/event.html#midi

MIDI Documentation Example

Run the above code in miniAudicle. It's called midiIn.ck and it's in the m208Lab13 folder.

```
[chuck](VM): sporking incoming shred: 1 (midiDocExample.ck)...
144 36 88
128 36 64
144 38 97
128 38 64
144 40 84
128 40 64
144 42 112
128 42 64
```

144 decimal is 90 hex and 128 decimal is 80 hex. These are the staus bytes for a note on (NON) and a note off (NOF).

Problems

If you don't see anything MIDI message printouts in the Console Monitor maybe you're trying to connect to an invalid MIDI device number. On my computer the nanoPAD2 was input device 1, not 0. Change this line and run the program again.

```
// open midi receiver, exit on fail
if ( !min.open( 1 ) ) me.exit();
```

Strike the touch pads and move your finger around the XY touch pad.

```
[chuck](VM): sporking incoming shred: 1 (midiExample.ck)...
144 59 6
128 59 1
144 63 20
128 63 2
144 65 12
128 65 4
144 66 28
128 66 3
```

Remove all shreds. Save the file. Quit and restart miniAudicle. Restarting miniAudicle may not always be necessary, but if your program stops responding to MIDI input that's the first thing to try.

MIDI Messages

The majority of MIDI messages consist of three numbers called the status byte, data 1 byte, and data 2 byte. Chuck calls them msg.data1, msg.data2, and msg.data3. All three numbers are 8 bit numbers that range from 0 to 255 ($2^8 - 1$). Status bytes range from 128 – 255 (80-FF hex) and data bytes range from 0 – 127 (0-7F hex).

Status byte messages are divided into eight categories, the most common being Note Off, Note On, Control, Program Change, and Pitch Bend messages. Each of the eight status categories can send messages to 16 different MIDI channels independently. Each of the 16 channels can play a different instrument.

The output from the first example program displayed the MIDI status bytes in decimal format. It is MUCH easier to decipher MIDI messages when the status byte is displayed in the hexadecimal (base 16) number system. Create this program and you'll see what I mean.

byte2hex.ck

```
// byte2hex.ck
// John Ellinger Music 208 Winter2014
// converts a MIDI status byte to hexadecimal display

function string byte2hex( int num )
{
    // in a two digit hex byte each digit is called a nibble
    num / 16 => int msn; // most significant nibble, digit on left
    num % 16 => int lsn; // least significant nibble, digit on right

    ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
    "A", "B", "C", "D", "E", "F"] @=> string h[];
    return h[ msn ] + h[ lsn ];
}

// range of MIDI status bytes
```

```
<<< "==== msg.data1 STATUS BYTES in hex", "" >>>;
<<< byte2hex( 128 ), "-", byte2hex( 143 ), "is NOTE OFF" >>>;
<<< byte2hex( 144 ), "-", byte2hex( 159 ), "NOTE ON" >>>;
<<< byte2hex( 160 ), "-", byte2hex( 175 ), "AFTER TOUCH POLY" >>>;
<<< byte2hex( 176 ), "-", byte2hex( 191 ), "CONTROL MESSAGE" >>>;
<<< byte2hex( 192 ), "-", byte2hex( 207 ), "PROGRAM CHANGE" >>>;
<<< byte2hex( 208 ), "-", byte2hex( 223 ), "AFTER TOUCH CHANNEL" >>>;
<<< byte2hex( 224 ), "-", byte2hex( 239 ), "PITCH BEND" >>>;
<<< byte2hex( 240 ), "-", byte2hex( 255 ), "SYSTEM EXCLUSIVE" >>>;
// range of MIDI data bytes
<<< "\n==== msg.data2 and msg.data3, DATA BYTES in hex", "" >>>;
<<< byte2hex( 0 ), "is MINIMUM data byte in hex is 0 decimal" >>>;
<<< byte2hex( 127 ), "is MAXIMUM data byte in hex is 127 decimal" >>>;
```

You should see these results in the Console Window.

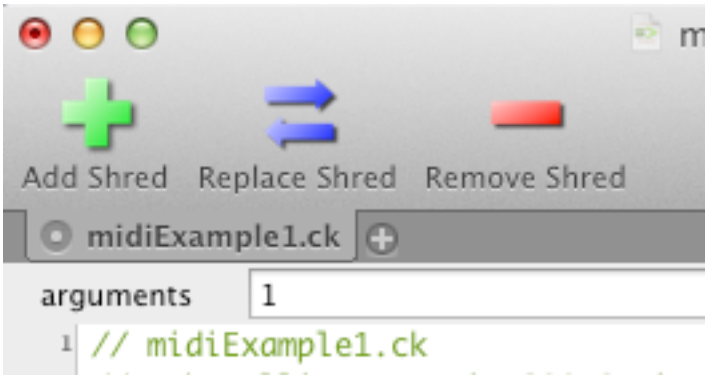
```
[chuck](VM): sporking incoming shred: 1 (byte2hex.ck)...
==== msg.data1 STATUS BYTES in hex
80 - 8F is NOTE OFF
90 - 9F NOTE ON
A0 - AF AFTER TOUCH POLY
B0 - BF CONTROL MESSAGE
C0 - CF PROGRAM CHANGE
D0 - DF AFTER TOUCH CHANNEL
E0 - EF PITCH BEND
F0 - FF SYSTEM EXCLUSIVE

==== msg.data2 and msg.data3, DATA BYTES in hex
00 is MINIMUM data byte in hex is 0 decimal
7F is MAXIMUM data byte in hex is 127 decimal
```

midiInExample.ck

Save midiDocExample.ck as midiInExample.ck. Modify the opening lines to read the input args() and add the byte2hex() function. Then enter this code.

```
// midiInExample.ck
// John Ellinger Music 208 Spring2014
//  modified: http://chuck.cs.princeton.edu/doc/language/
event.html#midi

// Input device number to open (see: chuck --probe)
0 => int device;
// get alternative device from the command line
if( me.args() ) me.arg(0) => Std.atoi => device;

MidiIn min;  // the midi event
MidiMsg msg; // the message for retrieving data
// open the device
if( !min.open( device ) ) me.exit();
// print out device that was opened
<<< "MIDI device:", min.num(), " -> ", min.name() >>>;

function string byte2hex( int num )
{
    // in a two digit hex byte each digit is called a nibble
    num / 16 => int msn; // most significant nibble, digit on left
    num % 16 => int lsn; // least significant nibble, digit on right

    ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
    "A", "B", "C", "D", "E", "F"] @=> string h[];
    return h[ msn ] + h[ lsn ];
}

// never ending loop
while( true )
{
    // wait on the event 'min'
    min => now;
    // get the message(s)
    while( min.recv(msg) )
    {
        // print out midi message
        <<< byte2hex(msg.data1), msg.data2, msg.data3 >>>;
    }
}
```

Restart miniAudicle but don't run the code just yet. Instead of changing the MIDI Input device number in code, enter its number in miniAudicle's arguments text field.



If you're running midiInExample.ck from the Terminal, add :1 after .ck like this.

```
501,je@jemac:~/Desktop/m208Lab13/2014$ chuck midiInExample.ck:1
MIDI device: 1  ->  nanoPAD2 PAD
90 36 95
80 36 64
90 37 53
80 37 64
```

nanoPAD2 Controls

The nanoPAD2 has 16 velocity sensative touch pads and an XY control surface. Each of the 16 touch pads sends a Note On message when pressed and a Note Off message when released. The SCENE button has four states that control the range of MIDI notes that are sent. Scene 1 sends MIDI note numbers 36-51, scene 2 sends notes 52-67, scene 3 sends notes 68-83, and scene 4 sends notes 84-99.

The XY Touch Pad sends MIDI Control message data from 0-127 along each axis.

Verify the touch pad messages shown above for scene 1.

nanopadExample1.ck

nanopadExample2.ck uses the 16 touch pads and three scenes to create different result for each scene. Scene 1 is a 16 note minor blues scale. Scene 2 uses sampled farm animal sounds. Scene 3 is a 16 note MIDI drum kit.

```
// nanopadExample1.ck
// John Ellinger Music 208 Spring2014
// modified: http://chuck.stanford.edu/doc/examples/basic/fm2.ck

// number of the device to open (see: chuck --probe)
0 => int deviceIn;
0 => int deviceOut;
// get command line arguments
<<< me.args() >>>;
if( me.args() == 2 )
{
    me.arg(0) => Std.atoi => deviceIn;
    me.arg(1) => Std.atoi => deviceOut;
}
// the MIDI input event
MidiIn min;
// open the device
if( !min.open( deviceIn ) ) me.exit();
// print out device that was opened
<<< "MIDI In device:", min.num(), " -> ", min.name() >>>;

// the MIDI output event
MidiOut mout;
// open the device
if( !mout.open( deviceOut ) ) me.exit();
// print out device that was opened
```

```
<<< "MIDI Out device:", mout.num(), " -> ", mout.name() >>>;

// the message for retrieving data
MidiMsg mmsg;

// 16 pads, one note each
[ -5, -2, 0,  3,  5,  6,  7,  10,
  12, 15, 17, 18, 19, 22, 24, 27 ] @=> int minorBlues[];

[ 36, 38, 41, 42, 45, 46, 50, 51,
  52, 53, 55, 60, 61, 63, 64, 66  ] @=> int drumNotes[];

function string byte2hex( int num )
{
    num / 16 => int msb; // most significant byte
    num % 16 => int lsb; // least significant byte

    ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
    "A", "B", "C", "D", "E", "F"] @=> string h[];
    return h[ msb ] + h[ lsb ];
}

function int isNON( int data1 ) // check for Note On
{
    if ( data1 >= 0x90 && data1 <= 0x9f )
        return 1;
    else
        return 0;
}

function int isNOF( int data1 ) // check for Note Off
{
    if ( data1 >= 0x80 && data1 <= 0x8f )
        return 1;
    else
        return 0;
}

function int patchChange( int instrument ) // send patch change
{
    MidiMsg msg;
    0xC0 => msg.data1;
    instrument => msg.data2;
```

```
    0 => msg.data3;
    mout.send( msg );
}

function int isNanoPadScene1( int data2 )
{
    if ( data2 > 35 && data2 < 52 )
        return 1;
    else
        return 0;
}

function int isNanoPadScene2( int data2 )
{
    if ( data2 > 51 && data2 < 68 )
        return 1;
    else
        return 0;
}

function int isNanoPadScene3( int data2 )
{
    if ( data2 > 67 && data2 < 85 )
        return 1;
    else
        return 0;
}

SndBuf buf[8];

[
"cat.wav",
"cow.wav",
"dog.wav",
"duck.wav",
"pig.wav",
"pony.wav",
"rooster.wav",
"sheep.wav"
] @=> string wavNames[];

for ( 0 => int ix; ix < buf.cap(); ix++ )
{
```

```
    buf[ix] => dac;
    me.sourceDir() + "/animals/" + wavNames[ix] => buf[ix].read;
    buf[ix].samples() => buf[ix].pos;
}

// this playIt function is only used for scene 2
// ix is the SndBuf array index
function void playIt( int ix )
{
    <<< "scene 2 sample", wavNames[ix] >>>;
    // samples are in the animals folder
    0 => buf[ix].pos;
    buf[ix].length() => now;
}

// infinite time-loop
while( true )
{
    // wait on the event 'min'
    min => now;

    // get the message(s)
    while( min.recv(mmsg) )
    {
        // the || token is a logical OR
        if ( isNON( mmsg.data1 ) || isNOF( mmsg.data1 ) )
                <<< byte2hex(mmsg.data1), mmsg.data2, mmsg.data3 >>>;
        {
            // print out midi message
            if  ( isNanoPadScene1( mmsg.data2 ) )
            {
                patchChange( 19 );
                36 => int scene1start;
                minorBlues[mmsg.data2-scene1start]+48 => int note;
                note => mmsg.data2;
                mout.send( mmsg ); // echo to MIDI out
                <<< "scene 1 note", note >>>;
            }
            else if  ( isNanoPadScene2( mmsg.data2 ) )
            {
                if ( isNOF( mmsg.data1 ) )
                    continue;
                52 => int scene2start;
```

```
                if ( mmsg.data2 > 59 )
                    8 -=> mmsg.data2;
                spork ~ playIt( mmsg.data2 - scene2start );
            }
            else if ( isNanoPadScene3( mmsg.data2 ) )
            {
                68 => int scene3start;
                if ( isNON( mmsg.data1 ) )
                    0x99 => mmsg.data1; // drum channel Note On
                else if ( isNOF(mmsg.data1) )
                    0x89 => mmsg.data1;  // drum channel Note Off

                drumNotes[ mmsg.data2 - scene3start ] => int note;
                note => mmsg.data2;
                mout.send( mmsg ); // echo to MIDI out
                <<< "scene 1 note", mmsg.data2 >>>;
            }

        }
    }
}
```

nanopadExample2.ck

nanopadExample2.ck uses the XY touch pad to control a form of FM synthesis. A more detailed view of FM synthesis is found in the Gampad section of this lab.

```
// nanopadExample2.ck
// John Ellinger Music 208 Spring2014
// modified: http://chuck.stanford.edu/doc/examples/basic/fm2.ck

// number of the device to open (see: chuck --probe)
0 => int deviceIn;
// get command line
<<< me.args() >>>;
if( me.args() )
    me.arg(0) => Std.atoi => deviceIn;
// the MIDI input event
MidiIn min;
// open the device
if( !min.open( deviceIn ) ) me.exit();
// print out device that was opened
<<< "MIDI In device:", min.num(), " -> ", min.name() >>>;
```

```
// the message for retrieving data
MidiMsg mmsg;

function int isController( int data1 ) // MIDI note off status
{
    if ( data1 >= 0xB0 && data1 <= 0xBf )
        return 1;
    else
        return 0;
}

// FM synthesis
SinOsc modulator => SinOsc carrier => dac;
47 => modulator.freq;
600 => carrier.freq;
0 => modulator.gain;
0 => carrier.gain;
2 => carrier.sync;

0 => int lastX;
0 => int lastY;
// never ending loop
while( true )
{
    // wait on the event 'min'
    min => now;

    // get the message(s)
    while( min.recv(mmsg) )
    {
        if ( isController( mmsg.data1 ) )
        {
            // print out midi message
            if ( mmsg.data2 == 1 )
            {
                0.8 => carrier.gain;
                mmsg.data3 => modulator.freq;
                mmsg.data3 => lastX;
            }
            else if ( mmsg.data2 == 2 )
            {
                mmsg.data3 + 300 => modulator.gain;
                mmsg.data3 => lastY;
```

```
            }
            else if ( mmsg.data2 == 16 )
            {
                <<< "Finger Off", "" >>>;
                0 => modulator.gain;
                0 => carrier.gain;
            }
            <<< "X,Y\t(", lastX, ",", lastY, ")" >>>;
        }
    }
}
```

MUSC 208 Winter 2014
John Ellinger Carleton College