Elliott Watson

RedID: 818846386

CompE 375:

Section 3 #21075

December 8, 2017

CompE 375 Embedded Systems Programming: Final Project Report: Generative MIDI sequencer

*Components*:

* DFrobot MIDI Arduino Shield

<https://www.dfrobot.com/wiki/index.php/Midi_shield_(SKU:DFR0157)>

* Arduino Uno (Without Arudino Libraries, only <avr/x.h> libraries and c standard library): Atmega328P development board

*IDE:* Atmel Studio 7.0

* exernal tool commands:
  + “C:\Program Files (x86)\arduino-nightly\hardware\tools\avr\bin\avrdude.exe”
  + “-C "C:\Program Files (x86)\arduino-nightly\hardware\tools\avr\etc\avrdude.conf" -p atmega328p -c Arduino -P COM9 -b 115200 -U flash:w:"$(ProjectDir)Debug\$(TargetName).hex":I”

*Peripherals*: Korg Volca FM digital synthesizer (MIDI in), MIDI cable, audio cable, speakers.

*Project dedicated hours:* 8 hours of research, exploration, and valuation; 12 hours of development and testing.

**Synopsis: Generative Midi Sequencer**

The basis of my project uses knobs and buttons of the DFrobot MIDI shield to guide probability based MIDI compositions as MIDI output data to some external MIDI controlled synthesizer or drum machine. The microcontroller acts as a probabilistic generating hub, where the user input of changing the values of the knobs and pressing the buttons have obvious effects on the looping but generative note structure.

The embedded software makes use of the Atmega328P’s UART Serial via 5 pin DIN MIDI cable. MIDI uses a baud rate of 31250. Additionally, the 328’s ADC is used to handle DFrobot’s analog potentiometers. Timer1 and an interrupt are used to trigger the MIDI outputs based on generated interval times, corresponding to a changing OCR1A as the max for the timer/counter.

The structure of MIDI output messages follow the above function definition:

void Midi\_Send(byte cmd, byte data1, byte data2) {

USART\_Transmit(cmd);

USART\_Transmit(data1);

USART\_Transmit(data2);

}

Where cmd is the MIDI instruction, and the data bytes (unsigned char) are specific MIDI parameters. For example, observe this call:

Midi\_Send(0x90,cMinor[i],0x45);

In this case, 0x90 is a *note on* message, cMinor[i] is giving a randomly selected value from a scale table array (a possible approach to generative composition), and the 0x45 is a value for the velocity of the note).

The heart of the generative functionality of the sequencer is based on Gen\_Rand(x,y), a wrapper function for the C language pseudo-random rand() function that takes a min and max argument.

**Program Flow (pseudo code):**

Run forever:

if Button 1 is pushed: start sequencer; if pressed again stop sequencer.

Sequencer uses Gen\_Rand() to determine the interval between notes and other midi outputs.

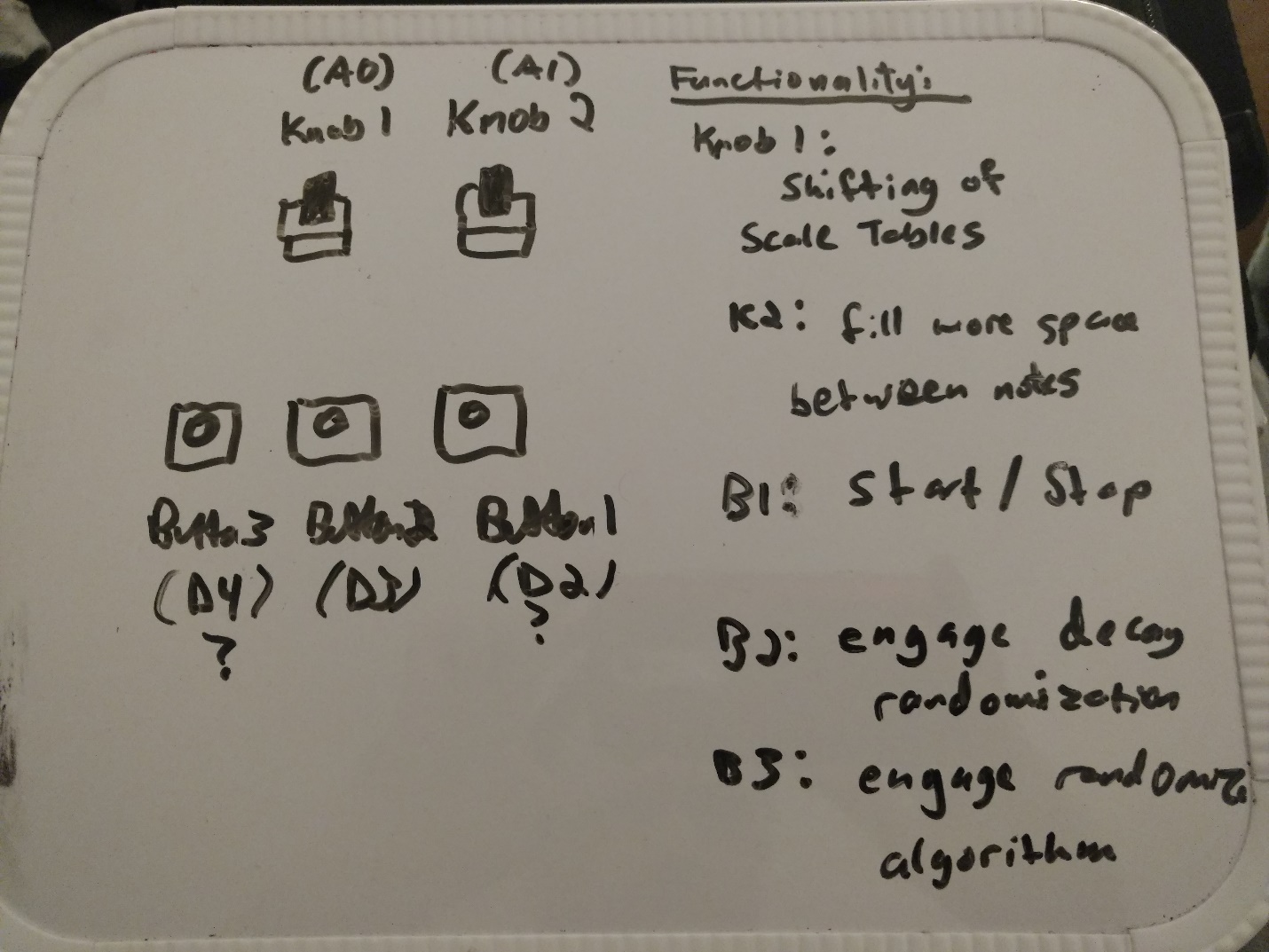
if Button 2 is pushed, enable/disable Modulator and Carrier Decay randomization.

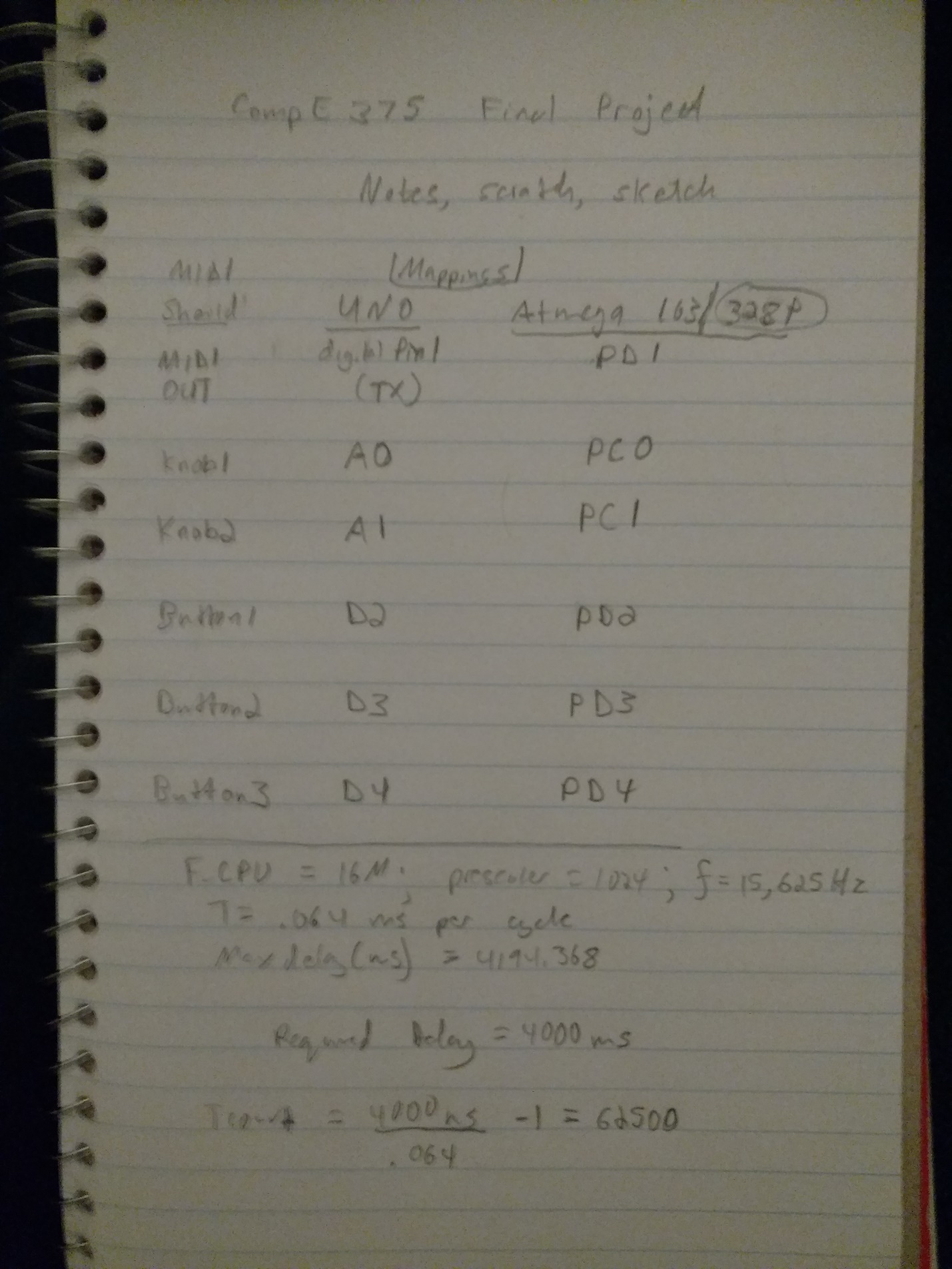
if Button 3 is pushed, enable/disable FM Algorithm randomization

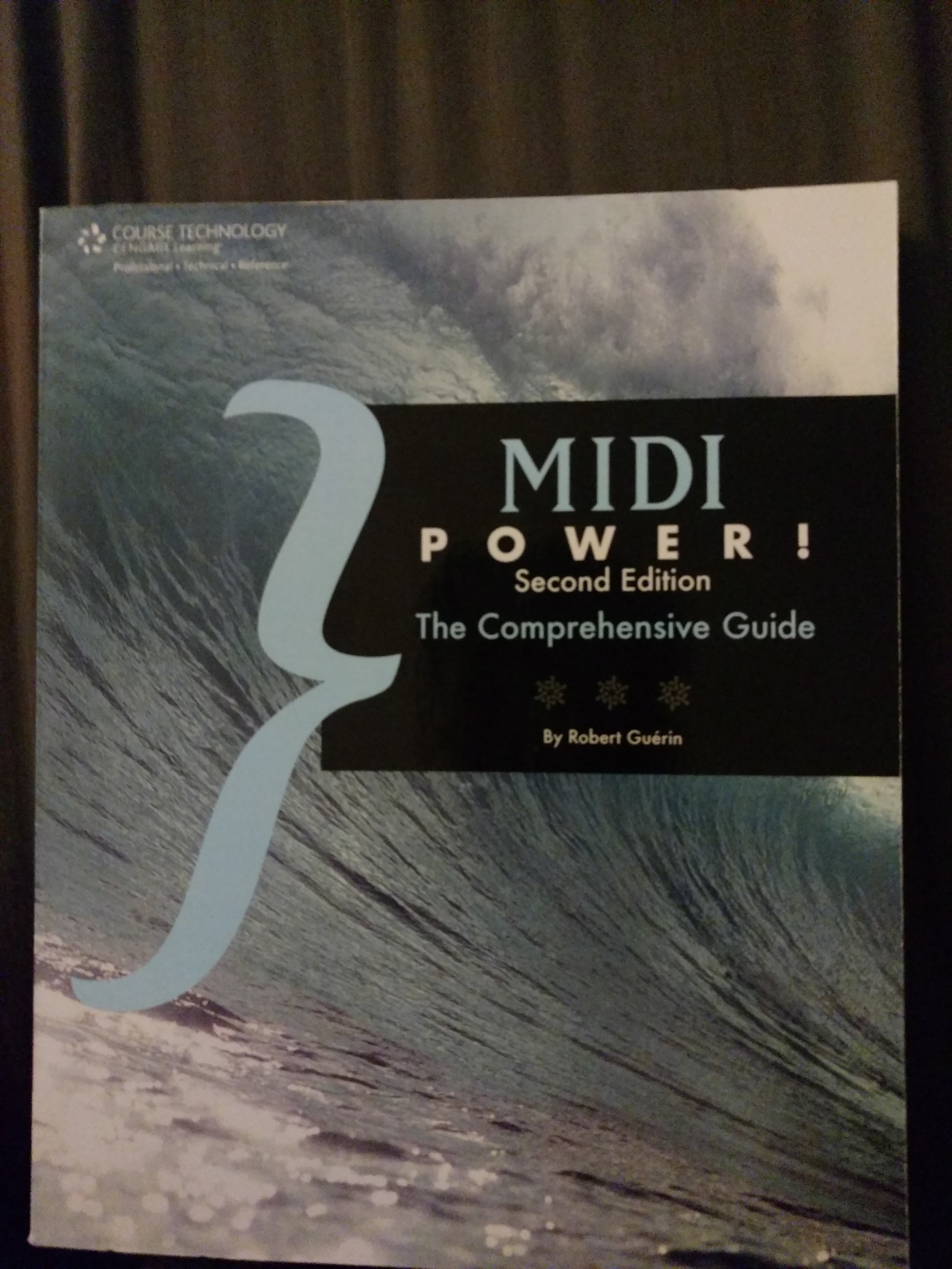
ADC read knob1: Time interval multiplier is based on 1 of 8 values (1024/128) to increase the duration scale of the MIDI message intervals (apparent rhythm)

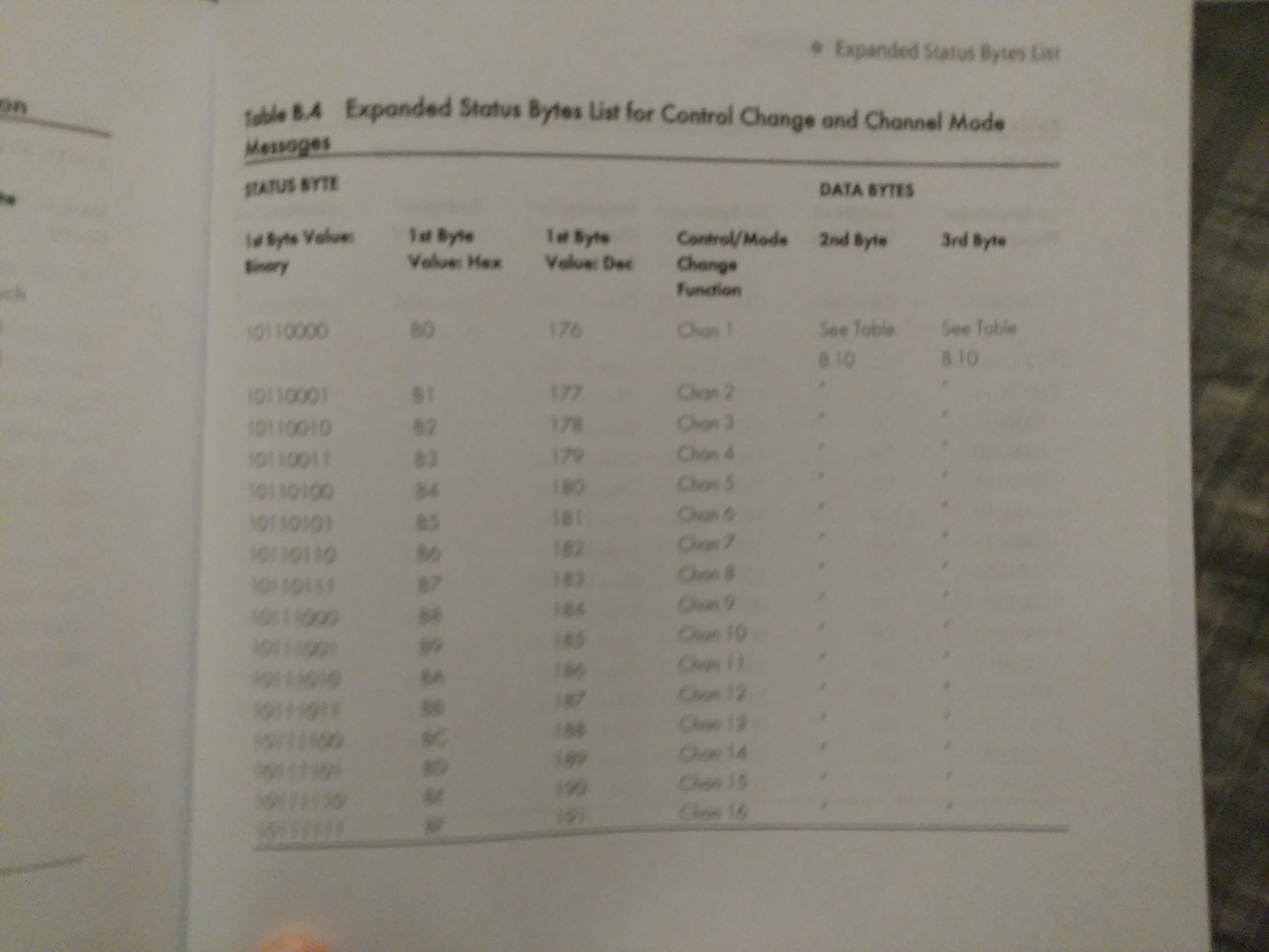
ADC read knob 2: Knob resolution is divided into 3 positions via if/else branching, determining whether the scale/note space uses C Minor scale, All C notes, or FMajor7 chord.

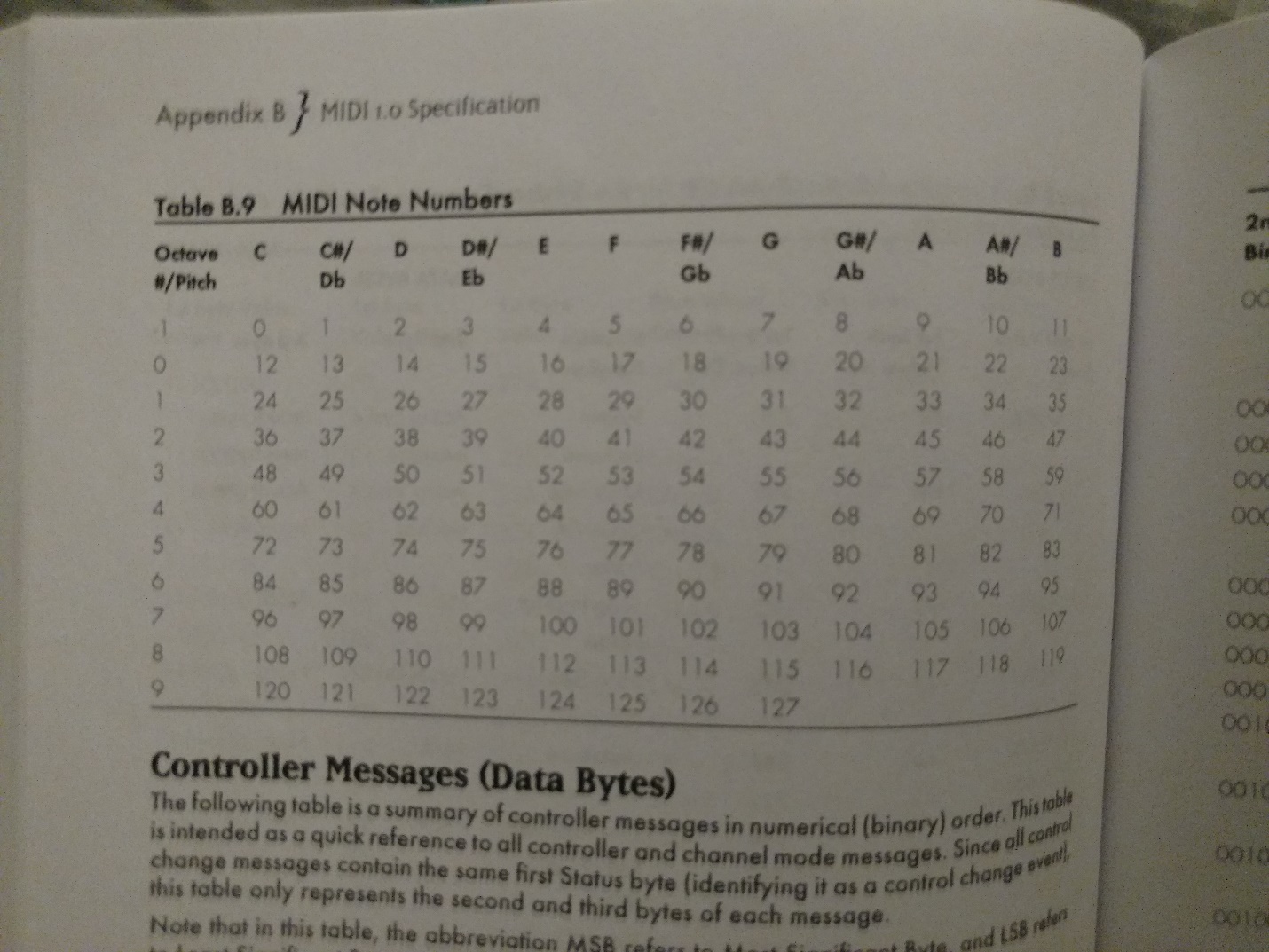
**Supporting Diagrams, Notes, and Documents:**

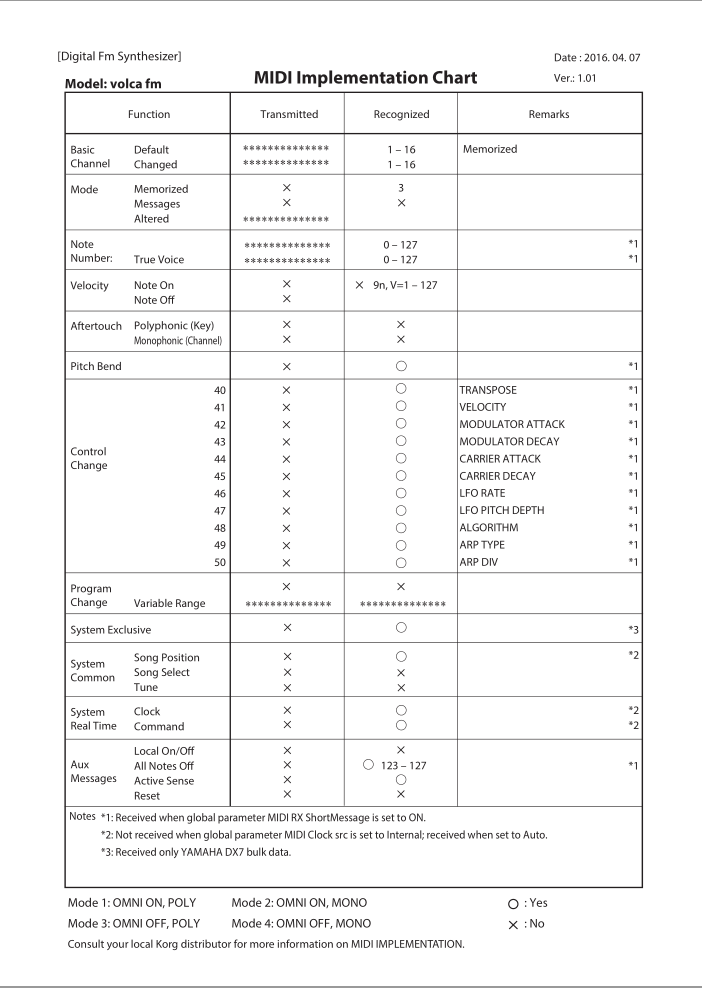
****

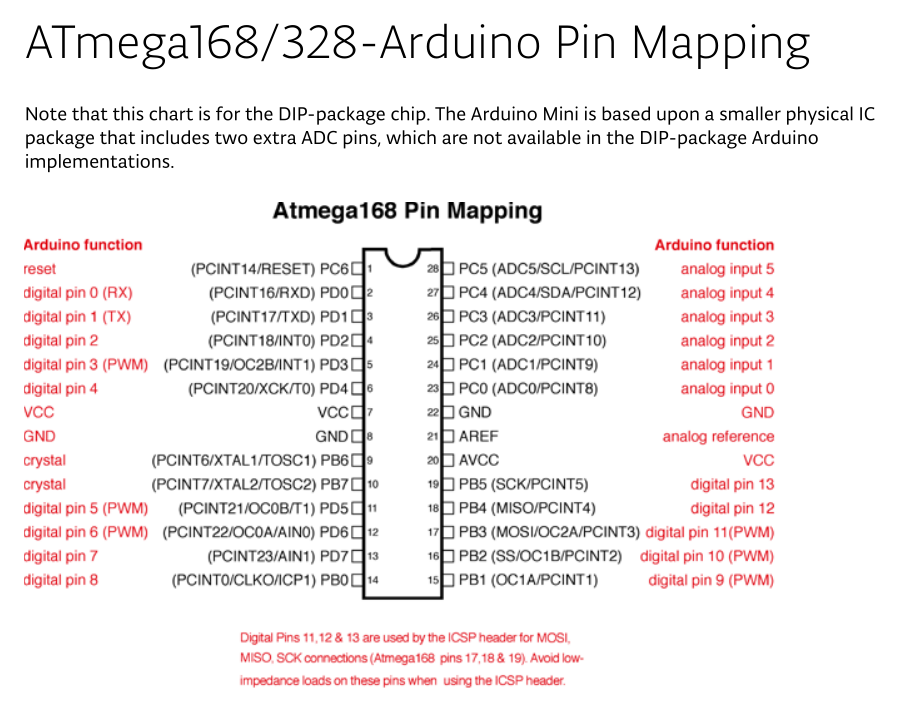
****

****

****

****

****

****

**Known issues:**

* Debouncing is weakly achieved using \_delay\_ms(); should be reimplemented with timer0 and extensively tested for ideal debounce time with mixed signal analyzer.

**Possible Functionality Expansion; Additional Features**

* This project provides a framework/basis for an expanded generative MIDI sequencer solution, where instead of the MIDI shield knob/switch inputs and actual MIDI controller could be used, and deeper more nuanced generative techniques can be combined with pre-determined compositional structure.

**Source Code:**

/\*

\* ElliottWatson\_CompE375\_FinalProject.c

\*

\* Created: 12/6/2017 7:16:35 PM

\* Author : Elliott Watson enwav@Darkhorse (enwave@hotmail.com; elliottnicholaswatson@gmail.com)

\*

\* This program is a generative MIDI sequencer using UART Serial over 5 pin DIN MIDI cable,

\* 16 bit AVR Timer1 with an interrupt for interval based MIDI message ouput

\* along with functions for random number generation, and ADC for the two nobs.

\*

\*/

#include <avr/io.h>

#include <avr/interrupt.h>

#include <util/delay.h>

#include <stdlib.h>

#include <time.h>

//defines byte as unisgned 8-bit int

//typedef uint8\_t byte;

typedef unsigned char byte;

#define *F\_CPU* 16000000UL// 16MHz clock from the debug processor

//pg 230 Atmega328p datasheet

//#define FOSC 1843200 // Clock Speed

#define FOSC 16000000 // Clock Speed

#define BAUD 31250

#define MYUBRR FOSC/16/BAUD-1

// defines for MIDI Shield components only

#define KNOB1 PINC0

#define KNOB2 PINC1

#define BUTTON1 PIND2

#define BUTTON2 PIND3

#define BUTTON3 PIND4

//for the ISA interrupt

volatile byte flag;

//for Timer1, the core timer; constantly changing based on probablity

const int top = 500;

void USART\_Init(unsigned int);

void USART\_Transmit(byte);

//char button(char button\_num);

void Midi\_Send(byte cmd, byte data1, byte data2);

//void USART\_Transmit(unsigned char);

void KNOB\_PIN\_Init();

int Rand\_Gen(int min, int max);

void adc\_Init();

int Knob\_Read(*uint8\_t* pin);

void timer\_Init();

//This interrupt enables the next set of MIDI output messages

ISR(TIMER1\_COMPA\_vect){

flag = 0x1;

}

int main(void)

{

//intializations

USART\_Init(MYUBRR);

KNOB\_PIN\_Init();

adc\_Init();

timer\_Init();

*srand*(*time*(*NULL*));

OCR1A = top;

//byte note = 45;

byte note, note2;

byte velocity;

byte keymod, tempomod;

byte decayFlag = 0x0, algFlag = 0x0;

int pot1, pot2;

int rndNoteIndex;

int rndDelay;

int rndMultOctave;

int rndMultSpace;

int rndModulatorDecay, rndCarrierDecay;

int rndAlg;

//int spaceMult = 1;

//int spaceMax;

const int cMinor[9] = {24,24,26,27,29,31,32,34,36};

const int e\_c[9] = {36,36,36,36,36,36,36,36,36};

const int fMajor7[9] = {17,17,21,24,28,29,33,36,40};

while (1)

{

pot1 = Knob\_Read(KNOB1);

keymod = pot1/8; // convert value to value 0-127

pot2 = Knob\_Read(KNOB2);

tempomod = pot2/128 + 1;

if(!(PIND & (1 << BUTTON1)))

{

*\_delay\_ms*(3000); //debounce; replace with timer0 setup

sei();

do{

//if(!(PIND & (1 << BUTTON2)))

//decayFlag = ~decayFlag;

//Knob reads and generative variable value updates

pot1 = Knob\_Read(KNOB1);

keymod = pot1/8; // convert value to value 0-127

pot2 = Knob\_Read(KNOB2);

tempomod = pot2/128 + 1;

rndNoteIndex = Rand\_Gen(0,8);

rndMultOctave = Rand\_Gen(0,2);

rndMultSpace = Rand\_Gen(1,4);

rndModulatorDecay = Rand\_Gen(0,127);

rndCarrierDecay = Rand\_Gen(60,120);

rndAlg = Rand\_Gen(1,127);

//Timer count is modulated to determine MIDI message spacing

OCR1A = top \* rndMultSpace \* tempomod;

//Note is determined from Knob\_read(keymod), random aray index, and chance for octave multiplier

if (keymod < 25)

note = cMinor[rndNoteIndex] + (12 \* rndMultOctave);

else if (keymod < 100)

note = e\_c[rndNoteIndex] + (12 \* rndMultOctave);

else

note = fMajor7[rndNoteIndex] + (12 \* rndMultOctave);

//Waveform decay randomization

if(decayFlag > 0){

Midi\_Send(0xB0,43, rndModulatorDecay);

Midi\_Send(0xB0,45, rndCarrierDecay);//carrier decay here

}

//FM algorithm randomization

if(algFlag > 0)

Midi\_Send(0xB0,48, rndAlg);

//MIDI note on

Midi\_Send(0x90, note, 0x45);

while(flag != 0x1){

if(!(PIND & (1 << BUTTON2)))

decayFlag = ~decayFlag;

if(!(PIND & (1 << BUTTON3)))

algFlag = ~algFlag;

}

Midi\_Send(0x80, note, 0x45);//note off

flag = 0x0;

}while((PIND & (1 << BUTTON1)));

cli();

*\_delay\_ms*(3000); //debounce, replace with Timer0 if possible

}

}

}

//pg 230 328p datasheet

//Initializes USART Registers for Serial I/O

//using 8-bit, 1 stop bit;

void USART\_Init( unsigned int ubrr)

{

/\*Set baud rate \*/

UBRR0H = (unsigned char)(ubrr>>8);

UBRR0L = (unsigned char)ubrr;

/\*Enable receiver and transmitter \*/

UCSR0B = (1<<RXEN0)|(1<<TXEN0);

/\* Set frame format: 8-bit data, 1 stop bit \*/

UCSR0C = (0<<USBS0)|(3<<UCSZ00);

}

void USART\_Transmit(byte data )

{

/\* Wait for empty transmit buffer \*/

while ( !( UCSR0A & (1<<UDRE0)) )

;

/\* Put data into buffer, sends the data \*/

UDR0 = data;

}

void Midi\_Send(byte cmd, byte data1, byte data2) {

USART\_Transmit(cmd);

USART\_Transmit(data1);

USART\_Transmit(data2);

}

void KNOB\_PIN\_Init(){

//data direction for buttons

DDRD |= (0 << DDD2);

DDRD |= (0 << DDD3);

DDRD |= (0 << DDD4);

//enable pull up resistors! Super important, didn't work until I did this!

PIND |= (1 << PIND2);

PIND |= (1 << PIND3);

PIND |= (1 << PIND4);

}

int Rand\_Gen(int min, int max){

int diff;

diff = max - min;

return *rand*() % diff + min;

}

void adc\_Init(){

//Reference voltage AREF = AVcc

ADMUX = (1 << REFS0);

//Using prescaler of 128: 16M/128 = 125000

ADCSRA = (1 << ADEN)|(1<<ADPS2)|(1<<ADPS2)|(1<<ADPS0);

}

int Knob\_Read(*uint8\_t* pin){

pin &= 0b00000111; //just makes sure the channel stays between 0-7

//clears bottom 3

ADMUX = (ADMUX & 0xF8)|pin;

//starts conversion

ADCSRA |= (1 << ADSC);

//waits for conversion to complete; ADSC becomes zero

while(ADCSRA & (1<<ADSC));

return(ADC);

}

void timer\_Init(){

//CTC mode; 1024 prescaler

TCCR1B |= (1 << WGM12)|(1 << CS12)|(1 << CS10);

TIMSK1 |= (1 << OCIE1A);

}