USB-MIDI-styrenhet

Projektrapport

Shadi Abdu-Rahman
2011-05-22

Abstract

This report describes the construction of a USB Midi Controller using an AVR ATmega16 microprocessor for processing and a UM232R USB-Serial Module for USB communication. The report includes source code for the USB Midi Controller as well as a Java MIDI Server.

Innehållsförteckning

Introduktion

Midistyrenheten är tänkt att användas att styra olika parametrar i instrument- och effekttillägg i musiksequencerprogram.

Många nyare sequencerprogram innehåller funktionalitet för att kommunicera med midistyrenheter i båda riktningar, dels reagera på inkommande signaler och skicka återkoppling så att midistyrenheten i sin tur kan visa aktuella värden på parametrar.

En del sequencerprogram har även funktionalitet som avbildar en uppsättning kontroller till instrumentet eller effekten i fokus, ofta i grupper om åtta parametrar. Användarvänligheten ställer dock vissa krav på styrenheten. Vid byte av instrument eller parametergrupp bör kontrollerna uppdateras till de nya parametrarnas aktuella värden, så att det inte sker ett plötsligt ”hopp” till det nya värdet när reglaget eller kontrollen justeras. Om ett reglage t.ex. ändrar en parameter A till värdet 32 innan reglaget ändras till styra en parameter B som vid byttillfället har värdet 64, vill man inte att reglaget ska skicka värdena 33, 34 osv när den justeras uppåt utan värdena 65, 66 osv.

Det finns olika sätt att lösa den här problematiken på. En del sequencerprogram tillämpar s.k. ”value pickup”. I exemplet ovan ändras inte parameterns värde förrän det fysiska reglaget nått värdet 64. Dock är den här lösningen problematisk. I exemplet ovan fungerar det så länge man vill höja parameter B:s värde medan en sänkning av densamma innebär att man som användare först måste långsamt vrida reglaget uppåt till 64 för att sedan sänka det, något som kan vara utmanande att hålla reda på vid t.ex. ett uppträdande.

Den lösning som implementerats här är ändlösa (360 grader) fysiska reglage och visuell indikation om vad den nya parametern har för värde på en LCD-display. De ändlösa reglagen skickar endast relativa värdeändringar medan absoluta parametervärden visas på LCD-displayen.

Midistyrenheten består av fyra ändlösa reglage, en LCD-display för visning av återkopplad information från sequencerprogram och fyra knappar för diverse funktionalitet.

Kravspecifikation

Följande är den ursprungliga kravspecifikationen med information om vilka krav som är implementerade:

Funktionella krav

Implementerade:

Krav F01: Varje reglage ska skicka information om sin relativa positionsändring till ett musiksequencerprogram.

Krav F02: Information om aktuella värden på de parametrar som styrs i ett musiksequencerprogram ska visas på en display.

Inte implementerade:

Krav F03: Det ska gå att navigera fokus bland instrument/effekter i de sequencerprogram som har funktioner för det.

Krav F04: Det ska gå att ändra uppsättningen parametrar som styrs i instrument/effekter i grupper om fyra i de sequencerprogram som har funktioner för det.

Krav F05: Aktuella instrument- och parameternamn ska visas på displayen för de sequencerprogram som tillhandahåller sådan information.

Icke-funktionella krav

Implementerade:

Krav IF01: Midistyrenheten ska kommunicera med en värddator via serieport eller, i mån av tid, via USB-port.

Krav IF02: Om USB-port används för kommunikation ska midistyrenheten drivas med ström från den.

Krav IF03: Sändning och mottagning av information ska ske med en sådan hastighet att ändringen upplevs som omedelbar, < 10 ms, av användaren.

Utförande

Komponentlista

1 AVR Atmega16-mikroprocessor

1 UM232R USB-RS232-omvandlarmodul

1 16x2 LCD (HD44780-kompatibel)

4 ändlösa reglage

4 momentära knappar sp-no

1 NTX 74HC165-skiftregister

1 grön lysdiod

1 röd lysdiod

lämpliga motstånd, kondensatorer,

virkort

virtråd

Hårdvara

De olika komponenterna monterades på ett s.k. virkort och kopplades ihop med virtråd. För bästa kontakt löddes stift för 0- och 5V längs med respektive räls på kortet. Rälsarna är i sin tur kopplade till USB-seriemodulens utgångar för 0- och 5V. På så sätt drivs enheten med ström från USB-porten den ansluts till.

LCDn kopplades in med ett 8-bitars gränssnitt till mikroprocessorn, knapparna direkt till varsin ingång på mikroprocessorn, medan reglagen kopplades till de parallella ingångarna på ett skiftregister vars

seriella utgång sedan kopplades ihop med mikroprocessorn. Detta för att spara ingångar på mikroprocessorn.

Programvaruimplementation

Avläsning av knappar och rattar

Knapparna används till att ändra den typ av parametrar som justeras av rattarna samt tillhörande visningsläge på LCD-displayen. De olika lägena är Device 1-4, Device 5-8, Volume 1-4 och Sends 1-4 för styrning av ett instruments parametrar 1-4, 5-8, kanalvolym 1-4 och kanalsänd 1-4 respektive.

Rattarna skickar information om riktningen de vridits. Denna information omvandlas sedan till Control Change MIDI-meddelanden.

Knapparna och rattarna avläses med ett timeravbrott med frekvensen 1 kHz, dvs. 1000 ggr/sekund.

Seriell kommunikation

Den seriella informationen hanteras på två sätt beroende på om den är inkommande eller utgående.

Inkommande MIDI-meddelanden från värddatorn utlöser ett hårdvaruavbrott lägger in dem i en ringbuffert för avkodning och visning på LCD-displayen när inget annat behöver göras.

Utgående MIDI-meddelanden som har sin källa då en ratt vrids skickas omgående till värddatorn så fort tillhörande timeravbrott utlöses. Detta då det är mer tidskritiskt att parameterändringar når musikprogrammet än att LCD-displayen uppdateras.

LCD-display

LCD-displayen uppdateras dels då en knapp tryckts ner för uppdatering av visningsläge och dels då det inkommit minst ett MIDI-meddelande.

Resultat

Midistyrenheten fungerar med de krav som är implementerade, men tiden räckte tyvärr inte till att implementera alla ursprungliga krav.

Även om enheten skickar och tar emot MIDI-meddelanden i rätt format behövdes en USB-MIDI-drivrutin på datorsidan. Det kändes som överkurs att implementera en sådan och istället programmerades en MIDI-server i Java som lyssnar på musikprogrammets utgående MIDI-port och lägger ut meddelandena på serieporten. Likaså lyssnar den på serieporten och lägger ut de meddelandena på musikprogrammets inkommande MIDI-port. Källkoden för denna återges i Bilaga 2.

Källförteckning

Under projektarbetet användes bl.a följande källor:

Wikipedia-artikel om Rotary Encoders:

http://en.wikipedia.org/wiki/Quadrature_encoder#Single-track_Gray_encoding

Datablad:

Atmel 74HC165 Microprocessor

FTDI UM232R USB- Serial UART Development Module

Hitachi HD44780U (LCD-II)

BOLYMIN BC1602A LCD

ALPS EC11 Encoder Series

NXP 74HC165 Shift Register

Bilaga 1 – C-källkod för mikroprocessor

MCU.c

```
#include <avr/io.h>
#include "defines.h"
#include <util/delay.h>
#include "model.h"
#include "usart.h"
#include "lcd.h"
#include "buttons.h"
#include "timer.h"

unsigned char inByte;
unsigned char inMessage[3];
int pos = 0;

int main()		{

			lcd_init();
			USART_init(12); //38400 bps @ 8MHz, error 0.2%
			sr_init();
			enc_init();
			buttons_init();
			timer_init();

			sei();

			while(1) {
					if (bytesToRead()) {
							inByte = recvByte();
							if (pos == 0) {
									if (inByte == 0xb0) {
											inMessage[pos] = inByte;
											pos = 1;
									}

							} else {
									inMessage[pos] = inByte;
									if (++pos == 3) {
											putInModelBuffer(inMessage);
											updateLCD();
											pos = 0;
									}
							}
					}
			}
			return 0;
}
```

defines.h

```
#ifndef DEFINES_H_
#define DEFINES_H_
```

```
#define F_CPU 8000000

#endif
```

model.h

```
#ifndef MODEL_H_
#define MODEL_H_

extern void setMode(unsigned char mode);
extern unsigned char getMode(void);
extern void putInModelBuffer(unsigned char* message);
extern unsigned char getModelBufferValue(int mode, int param);
extern void setModelBufferValue(int mode, int param, unsigned char value);

#endif
```

model.c

```
volatile unsigned char modeState = 0;
volatile unsigned char modelBuffer[4][4];


void setMode(unsigned char mode) {
		modeState = mode;
}
unsigned char getMode(void) {
		return modeState;
}

void putInModelBuffer(unsigned char* message) {
		unsigned char modeByte = message[1];
		if (modeByte < 4)
			modelBuffer[0][modeByte] = message[2];
		else if (modeByte < 8)
			modelBuffer[1][modeByte - 4] = message[2];
		else if (modeByte < 12)
			modelBuffer[2][modeByte - 8] = message[2];
		else if (modeByte < 16)
			modelBuffer[3][modeByte - 12] = message[2];
}

unsigned char getModelBufferValue(int mode, int param) {
		return modelBuffer[mode][param];
}

void setModelBufferValue(int mode, int param, unsigned char value) {
		modelBuffer[mode][param] = value;
}
```

usart.h

```
#ifndef USART_H_
#define USART_H_

#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>

#define TX_LED			PB0
#define RX_LED			PB1
#define BUFFERSIZE48

extern void USART_init(uint16_t ubrr);
extern void sendByte(unsigned char byte);
extern int bytesToRead(void);
extern unsigned char recvByte(void);

#endif
```

usart.c

```
#include "usart.h"

volatile unsigned char inBuffer[BUFFERSIZE];
volatile int inBufReadIt = 0, inBufWriteIt = 0;
volatile int inBytes = 0;

void USART_init(uint16_t ubrr) {
		//Set Baud rate
		UBRRH = (ubrr>>8);
		UBRRL = ubrr;

		//Using default format: Asynchronous mode, 8 bit, No parity, 1 stop bit

		//Enable RX Complete Interrupt, RX & TX
		UCSRB=(1<<RXCIE)|(1<<RXEN)|(1<<TXEN);

		//Configure TX and RX Led pins as outputs
		DDRB |= (1<<RX_LED)|(1<<TX_LED);
}

void sendByte(unsigned char byte) {
		PORTB |= (1<<TX_LED);
		while ((UCSRA & (1 << UDRE)) == 0);
		UDR = byte;
		PORTB &= ~(1<<TX_LED);
}

unsigned char recvByte(void) {
		char byte = inBuffer[inBufReadIt];
		if (++inBufReadIt == BUFFERSIZE)
				inBufReadIt = 0;
		--inBytes;
		return byte;
}

int bytesToRead(void) {
		return inBytes;
}

ISR(USART_RXC_vect) {
		PORTB |= (1<<RX_LED);
		inBuffer[inBufWriteIt] = UDR;
		if (++inBufWriteIt == BUFFERSIZE)
				inBufWriteIt = 0;
		++inBytes;
		PORTB &= ~(1<<RX_LED);
}
```

timer.h

```
#ifndef TIMER_H_
#define TIMER_H_

#include <avr/interrupt.h>
#include "buttons.h"
#include "encoders.h"

extern void timer_init(void);

#endif
```

timer.c

```
#include "timer.h"

void timer_init() {
		TCNT1 = 0x0000;
		TCCR1B = (1<<WGM12)|(1<<CS10); //No prescale = 8MHz
		OCR1A = 0x1f40; //Set Compare Register 1A to 8000 = 1 kHz
```

```
		TIMSK = (1<<OCIE1A); //Enable Comp1A Interrupt
}

ISR(TIMER1_COMPA_vect) {
		readEncoders();
		readButtons();
}
```

shiftreg.h

```
#ifndef SHIFTREG_H_
#define SHIFTREG_H_

#include <avr/io.h>
#include "defines.h"
#include <util/delay.h>
#include "lcd.h"

#define SR_PORT			PORTB
#define SR_DDR			DDRB
#define SR_PIN			PINB
#define SL				PINB6
#define CLK				PINB7
#define SER				PINB5
#define SER_IN			PB5

extern void sr_init(void);
extern unsigned char readSR(void);

#endif
```

shiftreg.c

```
#include "shiftreg.h"

void sr_init() {
		SR_DDR &= ~(1<<SER);
		SR_DDR |= (1<<SL)|(1<<CLK);

		SR_PORT &= ~(1<<CLK);
		SR_PORT &= ~(1<<SL);
		_delay_us(2);
}

unsigned char readSR() {
		unsigned char byte = 0;

		SR_PORT &= ~(1<<SL); //Load
		_delay_us(0.01);

		SR_PORT |= (1<<SL);				//Shift
		_delay_us(0.01);

		for (int i = 7; i >= 0; --i) {
			SR_PORT &= ~(1<<CLK);
			_delay_us(0.01);
			if ((SR_PIN & (1<<SER_IN)))
				byte |= 1 << i;
			SR_PORT |= (1<<CLK);
		}
		return byte;
}
```

encoders.h

```
#ifndef ENCODERS_H_
#define ENCODERS_H_

#include <avr/io.h>
#include "shiftreg.h"
#include "usart.h"
#include "model.h"
```

```
#include "lcd.h"

extern void enc_init(void);
extern void readEncoders(void);

#endif
```

encoders.c

```
#include "encoders.h"

volatile unsigned char encInput[4];
int states[16] = {0, 1, -1, 0, -1, 0, 0, 1, 1, 0, 0, -1, 0, -1, 1, 0};

void enc_init() {
		//Initialize encInput so that second read at startup doesn't result in ghost movement
		unsigned char firstRead = readSR();

		for (int i = 0; i < 4; ++i) {
			encInput[i]		|= (firstRead >> (i << 1)) & 0x03;
			encInput[i] = (encInput[i] << 2) & 0x0f;
		}
};

void handleEncoder(int enc, unsigned char index) {
		int direction = states[index];
		unsigned char value;
		unsigned char mode;
		if (direction) {
			mode = getMode();
			value = getModelBufferValue(mode, enc);
			if (direction == 1 && value < 127) {
				setModelBufferValue(mode, enc, ++value);
				sendByte(0xb0);
				sendByte((mode << 2) + enc);
				sendByte(64 + direction);
			} else if (direction == -1 && value > 0) {
				setModelBufferValue(mode, enc, --value);
				sendByte(0xb0);
				sendByte((mode << 2) + enc);
				sendByte(64 + direction);
			}
			updateLCD();
		}
}

void readEncoders() {
		unsigned char encoderStates = readSR();

		for (int i = 0; i < 4; ++i) {
			encInput[i]		|= (encoderStates >> (i << 1)) & 0x03;
			handleEncoder(i, encInput[i]);
			encInput[i] = (encInput[i] << 2) & 0x0f;
		}
}
```

buttons.h

```
#ifndef BUTTONS_H_
#define BUTTONS_H_

#include <avr/io.h>
#include "lcd.h"
#include "model.h"

#define DEV1		PD2
#define DEV2		PD3
#define VOL			PD6
#define SEND		PD7
```

```
extern void buttons_init(void);
extern void readButtons(void);

#endif
```

buttons.c

```
#include "buttons.h"

volatile unsigned char scannedState[4];

void buttons_init() {
		DDRD &= ~((1<<DEV1)|(1<<DEV2)|(1<<VOL)|(1<<SEND));
}

void readButtons() {
		scannedState[0] = (PIND & (1<<DEV1));
		scannedState[1] = (PIND & (1<<DEV2));
		scannedState[2] = (PIND & (1<<VOL));
		scannedState[3] = (PIND & (1<<SEND));

		for (short i = 0; i < 4; ++i) {
			if (scannedState[i] && i != getMode()) {
				setMode(i);
				updateLCD();
			}
		}
}
```

lcd.h

```
#ifndef LCD_H_
#define LCD_H_

#include <avr/io.h>
#include "defines.h"
#include <util/delay.h>
#include "model.h"

#define LCD_CTRL_PORT		PORTB
#define LCD_CTRL_DDR		DDRB
#define RS					PINB2
#define RW					PINB3
#define EN					PINB4
#define BF					PA7

#define LCD_DATA_PORT		PORTA
#define LCD_DATA_PIN		PINA
#define LCD_DATA_DDR		DDRA


extern void lcd_init(void);
extern void updateLCD(void);
extern void writeMode(unsigned char mode);
extern void writeValue(unsigned char value, unsigned char pos);
extern void setPos(unsigned char pos);
extern void clearLCD(void);
extern void writeLCD(unsigned char instr, unsigned char data);
extern void enableCycle(void);
extern void busyWait(void);

#endif
```

lcd.c

```
#include <stdlib.h>
#include "lcd.h"

char ones[1];
char tens[2];
```

```
char hundreds[3];

void writeRAW(unsigned char instr) {
		LCD_CTRL_PORT &= ~(1<<RS);
		LCD_CTRL_PORT &= ~(1<<RW);
		LCD_DATA_PORT = instr;
		_delay_us(0.1);
		enableCycle();
}

void lcd_init() {
		LCD_DATA_DDR = 0xff;
		LCD_CTRL_DDR |= (1<<EN)|(1<<RW)|(1<<RS);

		LCD_CTRL_PORT &= ~((1<<EN)|(1<<RW)|(1<<RS));
		LCD_DATA_PORT = 0x00;

		_delay_ms(15);
		writeRAW(0x30);
		_delay_ms(5);
		writeRAW(0x30);
		_delay_us(50);
		writeRAW(0x30);
		_delay_us(50);

		writeLCD(0x38, 0x00);
		writeLCD(0x0c, 0x00);
		writeLCD(0x01, 0x00);
		writeLCD(0x06, 0x00);

		updateLCD();
}

void updateLCD() {
		unsigned char mode = getMode();
		clearLCD();
		writeMode(mode);
		for (int i = 0; i < 4; ++i) {
				writeValue(getModelBufferValue(mode, i), i);
		}
}

void writeValue(unsigned char value, unsigned char pos) {
		setPos(40 + (pos << 2));
		if (value < 10) {
				itoa((int) value, ones, 10);
				writeLCD(0x30, 0x01);
				writeLCD(0x30, 0x01);
				writeLCD(ones[0], 0x01);

		} else if (value < 100) {
				itoa((int) value, tens, 10);
				writeLCD(0x30, 0x01);
				writeLCD(tens[0], 0x01);
				writeLCD(tens[1], 0x01);

		} else {
				itoa((int) value, hundreds, 10);
				writeLCD(hundreds[0], 0x01);
				writeLCD(hundreds[1], 0x01);
				writeLCD(hundreds[2], 0x01);
		}
}

void writeMode(unsigned char mode) {
		setPos(3);
		switch(mode) {
				case 0:
						writeLCD(0x44, 0x01);
						writeLCD(0x65, 0x01);
						writeLCD(0x76, 0x01);
```

```
                                        writeLCD(0x69, 0x01);
                                        writeLCD(0x63, 0x01);
                                        writeLCD(0x65, 0x01);
                                        writeLCD(0x20, 0x01);
                                        writeLCD(0x31, 0x01);
                                        writeLCD(0xb0, 0x01);
                                        writeLCD(0x34, 0x01);
                                        break;

                                case 1:
                                        writeLCD(0x44, 0x01);
                                        writeLCD(0x65, 0x01);
                                        writeLCD(0x76, 0x01);
                                        writeLCD(0x69, 0x01);
                                        writeLCD(0x63, 0x01);
                                        writeLCD(0x65, 0x01);
                                        writeLCD(0x20, 0x01);
                                        writeLCD(0x35, 0x01);
                                        writeLCD(0xb0, 0x01);
                                        writeLCD(0x38, 0x01);
                                        break;

                                case 2:
                                        writeLCD(0x56, 0x01);
                                        writeLCD(0x6f, 0x01);
                                        writeLCD(0x6c, 0x01);
                                        writeLCD(0x75, 0x01);
                                        writeLCD(0x6d, 0x01);
                                        writeLCD(0x65, 0x01);
                                        writeLCD(0x20, 0x01);
                                        writeLCD(0x31, 0x01);
                                        writeLCD(0xb0, 0x01);
                                        writeLCD(0x34, 0x01);
                                        break;

                                case 3:
                                        writeLCD(0x53, 0x01);
                                        writeLCD(0x65, 0x01);
                                        writeLCD(0x6e, 0x01);
                                        writeLCD(0x64, 0x01);
                                        writeLCD(0x73, 0x01);
                                        writeLCD(0x20, 0x01);
                                        writeLCD(0x31, 0x01);
                                        writeLCD(0xb0, 0x01);
                                        writeLCD(0x34, 0x01);
                                        break;
                        }
}

void setPos(unsigned char pos) {
                        writeLCD(0x80 + pos, 0x00);
}

void clearLCD() {
                        writeLCD(0x01, 0x00);
}

void writeLCD(unsigned char instr, unsigned char data) {
                        busyWait();
                        if (data) {
                                        LCD_CTRL_PORT |= (1<<RS);
                        } else {
                                        LCD_CTRL_PORT &= ~(1<<RS);
                        }
                        LCD_CTRL_PORT &= ~(1<<RW);
                        LCD_DATA_PORT = instr;
                        _delay_us(0.1);
                        enableCycle();
}

unsigned char busy() {
```

```
	unsigned char bf = 0;
	LCD_CTRL_PORT &= ~(1<<RS);
	LCD_CTRL_PORT |= (1<<RW);
	_delay_us(0.1);
	LCD_CTRL_PORT |= (1<<EN);
	_delay_us(0.5);
	if (LCD_DATA_PIN & (1<<BF))
		bf = 1;
	LCD_CTRL_PORT &= ~(1<<EN);
	_delay_us(0.5);
	return bf;
}

void busyWait() {
	LCD_DATA_DDR = 0x00;
	while(busy());
	LCD_DATA_DDR = 0xff;
}

void enableCycle() {
	LCD_CTRL_PORT |= (1<<EN);
	_delay_us(0.5);
	LCD_CTRL_PORT &= ~(1<<EN);
}
```

Bilaga 2 – Java-källkod för MIDI-server

MIDIServer.java

```
import gnu.io.CommPortIdentifier;
import gnu.io.NoSuchPortException;
import gnu.io.PortInUseException;
import gnu.io.SerialPort;
import gnu.io.UnsupportedCommOperationException;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Scanner;
import java.util.concurrent.ArrayBlockingQueue;
import javax.sound.midi.MidiDevice;
import javax.sound.midi.MidiSystem;
import javax.sound.midi.MidiUnavailableException;
import javax.sound.midi.Receiver;
import javax.sound.midi.Transmitter;

public class MIDIServer {
	static String DEFAULT_COMPORT = "COM1";
	static String DEFAULT_IN_DEVICE = "MCU IN";
	static String DEFAULT_OUT_DEVICE = "MCU OUT";

	static CommPortIdentifier portId = null;
	static SerialPort serialPort;
	static OutputStream outputStream;
	static InputStream inputStream;

	static MidiDevice inDevice = null;
	static MidiDevice outDevice = null;
	static Transmitter trans = null;
	static Receiver recv = null;

	public static void main(String[] args) {

		String comPortId = DEFAULT_COMPORT;
		String inDeviceName = DEFAULT_IN_DEVICE;
		String outDeviceName = DEFAULT_OUT_DEVICE;

		setupSerialPort(comPortId);

		setupMIDI(inDeviceName, outDeviceName);
```

```
                ArrayBlockingQueue<byte[]> outBuffer = new ArrayBlockingQueue<byte[]>(128);

                new Thread(new SerialWriter(outputStream, outBuffer)).start();
                new Thread(new SerialReader(inputStream, recv)).start();
                trans.setReceiver(new OutgoingMidiHandler(outBuffer));

                System.out.println("Serial MIDI Communication Server Started ...");
                System.out.println("Enter any character to shut it down. ");
                Scanner input = new Scanner(System.in);
                while (!input.hasNext()) {

                }
                tearDownSerialPort();
                tearDownMIDI();
                System.exit(0);
        }

        private static void setupSerialPort(String comPortId) {
                boolean prompted = false;
                while (portId == null) {
                        try {
                                portId = CommPortIdentifier.getPortIdentifier(comPortId);
                        } catch (NoSuchPortException e) {
                                if (!prompted) {
                                        System.out.println("The unit is not
connected to port " + comPortId + ". Please connect it. ");
                                        prompted = true;
                                }
                                try {
                                        Thread.sleep(50);
                                } catch (InterruptedException ie) {}
                        }
                }

                if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL) {

                        try {
                                serialPort = (SerialPort) portId.open("MCU", 2000);
                        } catch (PortInUseException e) {
                                System.out.println("Error opening port. Port is in use.");
                                System.exit(-1);
                        }

                        try {
                                serialPort.setSerialPortParams(38400,
SerialPort.DATABITS_8, SerialPort.STOPBITS_1,           SerialPort.PARITY_NONE);
                                serialPort.setOutputBufferSize(48);
                                serialPort.setInputBufferSize(48);
                        } catch (UnsupportedCommOperationException e) {

        System.out.println("UnsupportedCommOperationException occured. ");
                                serialPort.close();
                                System.exit(-1);
                        }

                        try {
                                outputStream = serialPort.getOutputStream();
                        } catch (IOException e) {
                                System.out.println("Error getting output stream. ");
                                serialPort.close();
                                System.exit(-1);
                        }

                        try {
                                inputStream = serialPort.getInputStream();
                        } catch (IOException e) {
                                System.out.println("Error getting input stream. ");
                                serialPort.close();
                                System.exit(-1);
                        }
```

```
                                        try {
                                                serialPort.notifyOnOutputEmpty(true);
                                                serialPort.notifyOnDataAvailable(true);
                                        } catch (Exception e) {
                                                System.out.println("Error setting event notification. ");
                                                serialPort.close();
                                                System.exit(-1);
                                        }
                                } else {
                                        System.out.println(comPortId + " is not a serial port. ");
                                        System.exit(-1);
                                }
                        }

                        private static void tearDownSerialPort() {
                                try {
                                        if (inputStream != null)
                                                inputStream.close();
                                        if (outputStream != null)
                                                outputStream.close();
                                } catch (IOException e) {}
                                if (serialPort != null)
                                        serialPort.close();
                        }

                        private static void setupMIDI(String inDeviceName, String outDeviceName) {
                                MidiDevice.Info[] devices = MidiSystem.getMidiDeviceInfo();
                                boolean rxFound = false, txFound = false;
                                int i = 0;
                                while (i < devices.length && (!rxFound || !txFound)) {
                                        try {
                                                if (devices[i].getName().equals(inDeviceName)) {
                                                        inDevice =
MidiSystem.getMidiDevice(devices[i]);
                                                        if (inDevice.getMaxReceivers() == -1) {
                                                                inDevice.open();
                                                                recv =
inDevice.getReceiver();
                                                                rxFound = true;
                                                        }
                                                }
                                                else if (devices[i].getName().equals(outDeviceName)) {

                                                        outDevice =
MidiSystem.getMidiDevice(devices[i]);
                                                        if (outDevice.getMaxTransmitters() ==
-1) {
                                                                outDevice.open();
                                                                trans =
outDevice.getTransmitter();
                                                                txFound = true;
                                                        }
                                                }
                                        } catch (MidiUnavailableException e) {
                                                System.exit(-1);
                                        }
                                        i++;
                                }
                        }

                        private static void tearDownMIDI() {
                                inDevice.close();
                                outDevice.close();
                        }
}
```

OutgoingMidiHandler.java

```
import java.util.concurrent.ArrayBlockingQueue;
```

```
import javax.sound.midi.MidiMessage;
import javax.sound.midi.Receiver;
import javax.sound.midi.ShortMessage;


public class OutgoingMidiHandler implements Receiver {
		private ArrayBlockingQueue<byte[]> outBuffer;
		int messageCount = 0;

		public OutgoingMidiHandler(ArrayBlockingQueue<byte[]> outBuffer) {
			this.outBuffer = outBuffer;
		}

		public void send(MidiMessage message, long timeStamp) {
			produce(message);
		}

		private synchronized void produce(MidiMessage message) {
			byte[] byteMess = message.getMessage();
			if (byteMess[0] == (byte) ShortMessage.CONTROL_CHANGE) {
				try {
					outBuffer.add(byteMess);
				} catch(IllegalStateException e) {
					System.out.println("Outgoing Traffic Congestion. ");
					outBuffer.poll();
					outBuffer.offer(byteMess);
				}
			}
		}

		public void close () {
			outBuffer.clear();
		}
}
```

SerialWriter.java

```
import java.io.IOException;
import java.io.OutputStream;
import java.util.concurrent.ArrayBlockingQueue;


public class SerialWriter implements Runnable {
		private OutputStream outputStream;
		private ArrayBlockingQueue<byte[]> outBuffer;
		byte byteMess[] = new byte[3];
		int messageCount = 0;

		public SerialWriter(OutputStream outputStream, ArrayBlockingQueue<byte[]> outBuffer) {
			this.outputStream = outputStream;
			this.outBuffer = outBuffer;
		}

		public void run() {
			while(true) {
				consume();
			}
		}

		private synchronized void consume() {
			try {
				if (outBuffer.size() > 0) {
					byteMess = outBuffer.poll();
					outputStream.write(byteMess);
				} else {
					try {
						Thread.sleep(10);
					} catch (InterruptedException e) {}
				}
			} catch (IOException e) {
				System.out.println("Error writing to serial port. Terminating ...");
```

```
                                        System.exit(-1);
                                }
                        }
}
```

SerialReader.java

```
import java.io.IOException;
import java.io.InputStream;

import javax.sound.midi.InvalidMidiDataException;
import javax.sound.midi.Receiver;
import javax.sound.midi.ShortMessage;


public class SerialReader implements Runnable {
                        InputStream inputStream;
                        private Receiver recv;
                        int readByte;
                        byte byteMess[] = new byte[3];
                        int pos = 0;
                        int messageCount = 0;
                        ShortMessage message = new ShortMessage();

                        public SerialReader(InputStream inputStream, Receiver recv) {
                                this.inputStream = inputStream;
                                this.recv = recv;
                        }

                        public void run() {
                                while (true) {
                                        produce();
                                }
                        }

                        private synchronized void produce() {
                                try {
                                        while ((readByte = inputStream.read()) > -1) {
                                                if (readByte == (byte) ShortMessage.CONTROL_CHANGE)
                                                        pos = 0;
                                                byteMess[pos] = (byte) readByte;
                                                if (++pos == 3) {
                                                        pos = 0;
                                                        try {
                                                        message.setMessage((int)
byteMess[0], (int) byteMess[1], (int) byteMess[2]);
                                                        } catch (InvalidMidiDataException e) {}
                                                        recv.send(message, -1);

/*Resend message as a quick fix for a bug in either DAW software
* or Java MIDI API where every other message is ignored.*/
                                                        recv.send(message, -1);
                                                }
                                        }
                                        try {
                                                Thread.sleep(1);
                                        } catch (InterruptedException e) {}
                                } catch (IOException e) {
                                        System.out.println("Error reading from serial port. Terminating ...");
                                        System.exit(-1);
                                }
                        }
}
```