LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg

Philip Damberg

Emil Jönsson

Andreas Nilsson

Måns Prahl

Klockflöjten

En MIDI-kontroller i flöjtformat

Projekt

4,5 högskolepoäng

Bertil Lindvall, Lars-Göran Larsson

Digitala system

År 1

Sammanfattning

Detta projekt bestod av att tillverka en prototyp av en MIDI-kontroller i formen av en flöjt. Prototypen designades efter eget huvud och är kapabel att sända MIDI-meddelanden till en mottagare för att på så vis kunna användas som styrenhet för virtuella instrument.

Nyckelord

Atmega1284, mikroprocessor, programspråket C, MIDI, USART, blockflöjt, virtuella instrument.

Abstract

This project consisted of the creation of a MIDI-controller prototype in the shape of a recorder flute. The prototype was designed freely by the students working on the project and is capable of sending MIDI messages to a receiver in order to be used as a controller for virtual instruments.

Keywords

Atmega1284, microprocessor, C programing language MIDI, USART, recorder, virtual instruments.

Innehållsförteckning

Förord

Detta projekt grundades i vårt gemensamma musikintresse. Vi var alla väldigt säkra på att vi ville göra något musikrelaterat, men osäkra på just vad. Medarbetare Philip Damberg hade sett intressanta projekt på olika slags digitala musikinstrument på intressegrupper och nätföreningar för Atmel- och AVR-styrenheter. Gruppen var enig om att detta lät som en rolig samt relevant idé och började spåna på exakt hur det skulle kunna lösas. Det som redovisas härefter är resultatet av gruppens resa.

1 Inledning

1.1 Syfte

Syftet med projektet är att skapa en digital representation av en blockflöjt med vilken användaren kan kontrollera olika virtuella instrument.

1.2 Målformulering

Projektets mål är att skapa en fungerande digital flöjt med 11 knappar och en tryckmätare, samtliga kopplade till en MCU av modellen ATmega1284 [1].

1.3 Problemformulering

Tänkbara problem är hur knappstudsar ska hanteras. Ytterligare problem skulle kunna vara de avbrott som kan behövas för att skicka MIDI-signaler till den externa enheten. Eftersom projektet innefattar både mjuk- och hårdvara finns många potentiella felkällor som kan vara svåra att identifiera. Eftersom gruppens medlemmar inte tidigare använt sig av AD-omvandling med tryckmätare skulle detta också kunna orsaka eventuella problem, vilka behöver beaktas. Variationer i atmosfärstryck gör att tryckmätaren behöver kalibreras vid uppstart. Under denna process är det viktigt att tryckmätaren är opåverkad.

1.4 Motivering av projektet

En stor motivation till varför projektgruppen valde att göra ett MIDI-projekt var det delade intresset för musik och datorteknik. En av gruppens medlemmar hade sedan innan erfarenhet av MIDI-kontrollers och hade därför en uppfattning om hur en digital flöjt skulle kunna programmeras och konstrueras. I gruppen fanns även en flöjtist som kunde ge en insikt om hur en tänkbar knappsats skulle designas. Eftersom en MIDI-kontroller enbart skickar instruktioner om vilka toner som spelas, medför den stor flexibilitet för musiker. I flöjtens fall kan dess MIDI-meddelanden användas för att styra flera olika virtuella instrument, t.ex. syntar såväl som andra blås- och stråkinstrument. Möjligheten att kunna transponera gör även att en flöjtist snabbt och enkelt kan ändra tonarten och spela med de inövade fingersättningarna.

1.5 Avgränsningar

Prototypen ska innefatta en knappsats, ha möjlighet till AD-omvandling av en insignal och konstrueras så att den går att hålla i och spelas på som en klassisk blockflöjt. Prototypen ska även kunna transponera toner upp och ned via särskilda knappkombinationer.

Mikrostyrenheten kommer programmeras till att hantera instruktioner för monofonisk tonspelning med volymstyrka beroende på den analoga insignalens styrka. Enheten avses inte kunna hantera övriga instruktioner som MIDI-protokollet tillhandahåller.

2 Teknisk bakgrund

MIDI (Musical Instrument Digital Interface) är ett standardiserat kommunikationsprotokoll som används för att styra elektroniska musikinstrument, ljudeffekter, ljussättningar m.m. MIDI-protokollet används av i stort sett alla tillverkare av elektronisk musikutrustning. En så kallad MIDI-kontroller skickar inget ljud, utan enbart instruktioner till ett virtuellt instrument om vilka toner som skall spelas, samt med vilken styrka. Hur instruktionerna behandlas avgörs av de enskilda virtuella instrumenten.

MIDI-meddelanden kan ha olika längd, men de som används för flöjten består av tre bytes vardera. Denna rapport omfattar några av de så kallade Channel Voice-meddelandena, mer specifikt Note Off och Note On, se tabell 2.1 nedan. För ytterligare information, se [2].

Instruktion (byte 1)	**Data (byte 2 och 3)**	**Beskrivning**
1000nnnn	0kkkkkkk 0vvvvvvv	*Note Off.* Avslutar en not. kkkkkkk motsvarar notvärdet. vvvvvvv motsvarar anslaget.
1001nnnn	0kkkkkkk 0vvvvvvv	*Note On.* Påbörjar en not. kkkkkkk motsvarar notvärdet. vvvvvvv motsvarar anslaget.

Tabell 2.1. Aktuella MIDI-meddelanden ('nnnn' i instruktionsbyten är adressbitar, som möjliggör styrning av flera enskilda enheter från en MIDI-kontroller)

3 Metod

3.1 Hårdvara

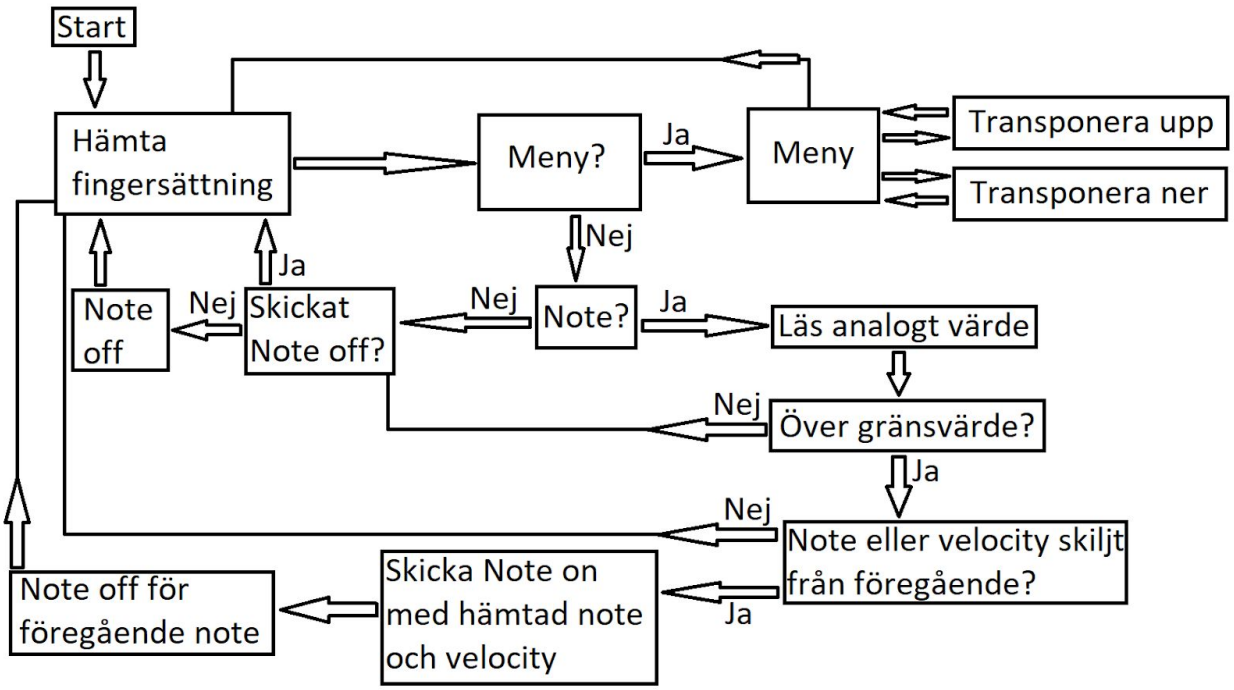
Klockflöjten skall med givna medel representera en blockflöjt spelmässigt. Detta görs genom att en serie knappar placeras på ett sätt som motsvarar hålen på en blockflöjt. En slang kopplad till en tryckmätare kommer användas för att registrera att flöjtisten blåser, vilket i kombination med fingersättningen genererar de spelinstruktioner vilka skall skickas som MIDI-meddelanden till det virtuella instrumentet. En analog tryckmätare [3] införskaffades för detta ändamål. Fingersättningen kopierades från en blockflöjt med 8 hål, av vilka 3 är "dubbelhål". Detta betyder att vi behöver totalt 11 knappar då dubbelhålen kräver två knappar var. Varje knapp kommer kopplas till var sitt ben på mikroprocessorn. Till knapparna används MCU:ns interna pullup-resistorer. För att likna en flöjt behöver MIDI-kontrollern programmeras till att bara

skicka en not åt gången, vilket innebär att en instruktion om att avsluta föregående not måste skickas i samband med att en annan not spelas. Till projektet behövs en operationsförstärkare för att förstärka spänningen som ges av tryckmätaren. Denna fungerar genom att den förstärker spänningen baserat på skillnaden mellan två valfria motstånd med hjälp av en matningsspänning från kortet på 5 V. Flöjten kommer ha en fingersättning som öppnar en meny, vilket indikeras av gul LED. Flöjtisten kommer i detta läge kunna transponera noterna upp och ned via knapparna. När fingersättningen för meny släpps, slocknar den gula LED:en och flöjten återgår till vanligt spelläge. Fingersättningar som inte motsvarar en ren ton kommer inte generera några MIDI-meddelanden.

3.2 Mjukvara

Programspråket C kommer användas i skapandet av mjukvarufunktionaliteten till flöjten. Assembly kommer eventuellt användas vid behov av avbrott när knappar trycks ned.

Under en konsert behöver samma dator ibland tillhandahålla flera virtuella instrument, vilka styrs av var sin MIDI-kontroller. För att undvika att meddelanden kolliderar är det viktigt att koden skrivs på ett sätt som gör att inga onödiga MIDI-meddelanden skickas. Klockflöjten memorerar den senast spelade noten och ser till att ett MIDI-meddelande skickas enbart då en förändring i not eller styrka har skett. En boolean i programkoden gör också att “Note off”-meddelanden inte skickas på nytt ifall det var det senast skickade meddelandet. Flera virtuella instrument har en inbyggd slide-effekt, vilket gör att instrumentet gör en steglös ändring till den nya noten istället för att tvärt byta not. För att kunna nyttja denna effekt skickar klockflöjten vid ny not först ett meddelande om att spela nya noten och därefter avsluta föregående. Detta gör ingen märkbar skillnad på de virtuella instrument som inte nyttjar denna effekt.



Figur 3.1. Flödesschema över programkoden

4 Analys

Skapandet av flöjten började med skriva metoder i programspråket C som skickar MIDI-meddelanden via mikroprocessorns USART-port till mottagande MIDI-enhet. Kännedom om hur detta gjordes skaffades genom att studera MIDI:s egna hemsida om vilka meddelanden som behövde skickas för att generera en ton samt stänga av denna. Mikroprocessorns bithastighet för USART behövde anpassas till 31250 bitar/sekund, vilket är den bithastighet som används av MIDI-protokollet. Därefter skrevs också metoder för att konvertera fingersättningen till de noter som motsvaras på en blockflöjt. Testmätning utfördes på ett kopplingsdäck med OP-förstärkare för tryckmätaren vid inget och vid fullt blås. Detta gav en skillnad på 1,5V vilket i teorin enkelt kan AD-omvandlas. MCU:n visade sig dock ge spridda värden vid AD-omvandling, varpå fler mätningar än förutsett blev nödvändiga för att skapa ett medelvärde och därigenom erhålla tillräcklig precision. Att göra fler AD-omvandlingar hade tagit längre tid, vilket hade gett

kännbara fördröjningar för flöjtisten. Att öka klockfrekvensen hade varit nödvändigt för att göra tillräckligt många avläsningar inom ett givet tidsintervall. På grund av tidsbrist och att ökad klockfrekvens skulle innebära omskrivningar av koden togs beslutet att montera en vridpotentiometer på kortet. Detta gav stabila analoga värden utan förstärkning. En vippströmställare installerades även för att kunna växla mellan vridpotentiometern och tryckmätaren som källa för analog insignal.

5 Resultat

Projektet resulterade i en prototyp med knappar ordnade efter barockgrepp. Knapparna låter användaren välja olika toner likt på en klassisk blockflöjt. En potentiometer ställs in och simulerar anblåsningen i den klassiska flöjten.

6 Slutsats

Tryckmätaren som använts var inte av lämplig typ för ändamålet. Knapparna visade sig sitta löst och upplevdes ömtåliga även om avläsning och MIDI-trafik fungerade felfritt. Att ha knappar och tryckmätare så tätt monterade gav ibland upphov till kortslutningar.

6.1 Framtida utvecklingsmöjligheter

Tack vare MIDI-protokollets enkla utförande är projektet skalbart. Protokollet innehåller metoder för att utöka det virtuella instrumentets styrning av *pitch bend* och *aftertouch* men även för att kunna kontrollera effekter hos mottagarenheten, så som *reverb-, flanger- och choruseffekter*.

Ett direkt utökningsprojekt vore att implementera en analog styrspak, en så kallad joystick, för att styra MIDI-meddelanden för pitch bend. På så vis kan en tagen ton enkelt böjas upp och ner.

Den befintliga hårdvarans generella kvalitet och utformning skulle behöva ses över innan eventuell serieproduktion träder i kraft.

7 Terminologi

Aftertouch: Möjlighet att i efterhand kunna modifiera intensiteten hos en spelad not.

Anslag: Den hastighet som exempelvis en pianotangent eller gitarrsträng slås an med.

MCU: Micro Controller Unit (mikroprocessor): I detta fall en AtMega 1284);

MIDI: Musical Instrument Digital Interface

Pitch bend: Steglös ändring av frekvens för toner.

Pullup-resistor: En i porten inbyggd resistor vilken tillhandahåller matningsspänning till benet i opåverkat läge. Yttre knappar förser benet med jordpotential.

Transponering: Ändrar samtliga noters tonhöjd.

USART: Universal asynchronous receiver-transmitter.

8 Källförteckning

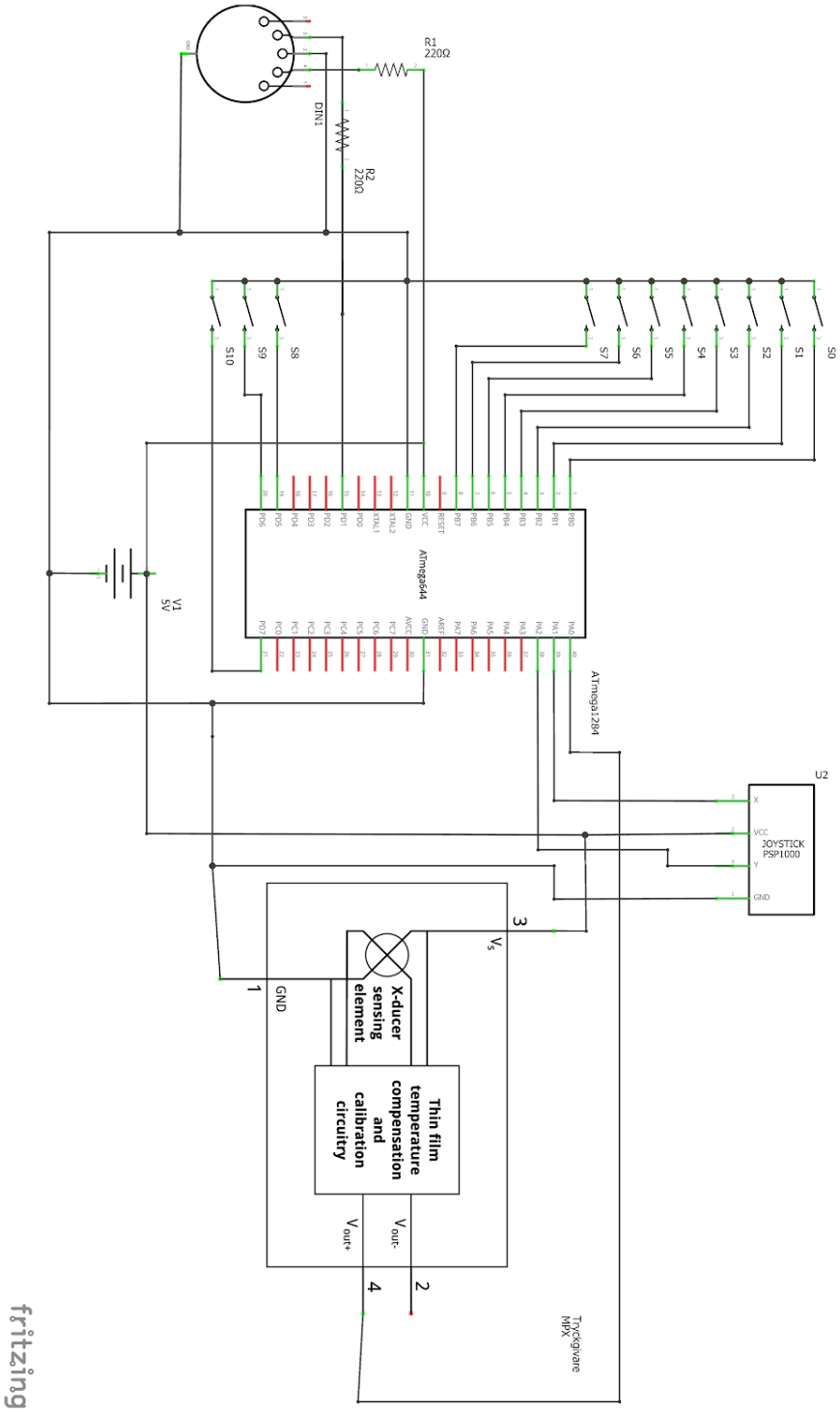
[1] Atmel, "ATmega1284 Datasheet Complete", ATmega1284 datablad. [Online]. Tillgänglig: http://ww1.microchip.com/downloads/en/devicedoc/atmel-42718-atmega1284_datasheet.pdf. [Reviderad okt 2016]

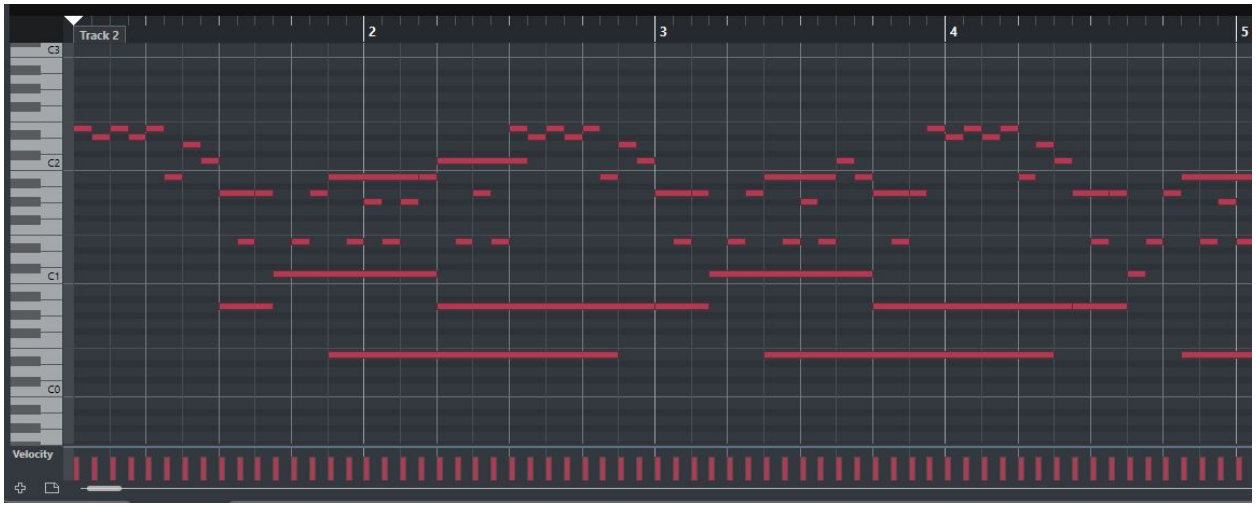
[2] MIDI Manufacturers Association, "Summary of Midi Messages", *Midi Manufacturers Association.* [Online]. Tillgänglig: https://www.midi.org/midi/specifications/item/table-1-summary-of-midi-message. [Hämtad: 20 maj, 2019]

[3] Honeywell Inc, "Pressure Sensors", 24PCFFM6G datablad [Online]. Tillgänglig: https://www.elfa.se/Web/Downloads/_t/ds/24PCxx_eng_tds.pdf [Hämtad 21 maj 2019]

9 Appendix

Figur 9.1. Kretsschema över prototypen med implementerad joystick. (roterad).





Figur 9.2 Exempel på MIDI-instruktioner för första takterna av Beethovens "Für Elise".

Programkod för klockflöjtens huvudprogram

```
/*
 * GccApplication1.c
 *
 * Created: 2019-04-03 12:45:31
 * Author : ph6108da-s
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include "global.h"
#include <util/delay.h>
#include "uart.h"
#include "midi.h"
#include <stdbool.h>



// Global attributes
uint8_t refNote;                    // Sets reference note to C4
uint8_t currNote; //Anger aktuell not
uint8_t prevNote; //En not som skickats görs om till
uint8_t currVel; //Anger aktuell velocity
uint8_t prevVel; //Anger föregåenge velocity
uint16_t fingering;
uint16_t menuFing = 1921; // ?0111 1000 0001?
uint8_t calibrationVel;
uint8_t currPres;
bool lastStop;          //Felipe mod 2k19

// Software related
void debugNote();
void stopAllNotes();
void startNote(uint8_t note, uint8_t vel);
void stopNote(uint8_t note);
```

```
void fluteMenu();
uint16_t checkFingering();
uint8_t convertFingeringToNote();
uint8_t getVelocity();
uint8_t convertADCtoVelocity();

// Hardware related
uint8_t getNote();
uint8_t getVelocity();
uint8_t calibrateVelocity();
void button_init();

int main(void)
{
	button_init();
	usart0_init(MIDI_BAUD_RATE);
	init_adc();
	refNote = 60;
	calibrationVel = 0;	// todo try make calibration, usually set to 0? :(
	currVel = 127;


   while (1) {

			fingering = checkFingering();

			if (fingering == menuFing) {
				fluteMenu();
			}
			else
			{
				currNote = convertFingeringToNote(fingering);
				currVel = getVelocity();
				if ((currNote != prevNote || currVel < calibrationVel) && (!lastStop)) {
					stopNote(prevNote);
					lastStop = true;
				}

				if (((currNote != prevNote))) { //Ifall note eller vel har ändrats från föregående samt att meny ej är valt - Skicka
not
					if(currNote != 255  && currVel > calibrationVel){
						startNote(currNote, currVel);
						stopNote(prevNote);
						lastStop = false;
					}else if (!lastStop) {
						stopAllNotes();
						lastStop = true;
					}
				}
				prevNote = currNote;
				prevVel = currVel;
			}
	}
}

void button_init(){
	PORTB = (0b11111111); //Aktiverar pull-up för ingångarna i B
	PORTD = (0b11100000); //Aktiverar pull-up för portarna 5, 6 och 7 i D
	DDRB = (0b00000000); //Anger att alla portar på B är ingångar
	DDRD &= ~(0b11100000); //Anger att portarna 5, 6 och 7 i D är ingångar (röd och gul LED på pin 2 och 3 utgångar )
	DDRD |= (0b00001100);
	PORTD |= (0b00001000); //Tänder röd lysdiod
}

void startNote(uint8_t note, uint8_t vel){
	usart0_transmit(MIDI_NOTE_ON);
```

```
	usart0_transmit(note);
	usart0_transmit(vel);  // TODO get velocity from potentiometer
}

void stopNote(uint8_t note){
	usart0_transmit(MIDI_NOTE_OFF);
	usart0_transmit(note);
	usart0_transmit(127);
}

void stopAllNotes(){
	usart0_transmit(0b10110000);
	usart0_transmit(0b01111011);
	usart0_transmit(127);
}

uint16_t checkFingering() { //Metod för att hämta fingersättningen
	uint8_t bFing = ~(PINB); //Inverterar fingersättningen på knapparna B (Nedtryckt knapp = 1)
	uint8_t dFing = ~(PIND); //Inverterar fingersättningen på knapparna D (Nedtryckt knapp = 1)
	dFing = dFing >> 5; //Bit-shiftar knapparna D så att 5-7 motsvarar 0-2.
	uint16_t allFing = dFing << 8; //Placerar D-knapparna på bitposition 8-10 i fingersättningen
	allFing = allFing + bFing; //Adderar B-knapparna till fingersättningen -> Alla knappar är i kronologisk ordning 0-10
	return allFing;
}

uint8_t convertFingeringToNote(uint16_t fingering){
	uint8_t note;
	switch (fingering){
		case 2047  :
		note  = refNote;  // C
		break;

		case 1919  :
		note = refNote+1; // Ciss
		break;

		case 1663  :
		note = refNote+2; // D
		break;

		case 1631  :
		note = refNote+3; // Diss
		break;

		case 1567  :
		note = refNote+4; // E
		break;

		case 2031  :
		note = refNote+5; // F
		break;

		case 1551  :
		note = refNote+6; // Fiss
		break;

		case 1543  :
		note  = refNote+7;  // G
		break;

		case 1563  :
		note = refNote+8; // Giss
		break;

		case 1539  :
		note = refNote+9; // A
```

```
break;

case 1549  :
note = refNote+10; // B
break;

case 1537  :
note = refNote+11; // H
break;

case 1538  :
note = refNote+12; // C
break;

case 3  :
note = refNote+13; // Ciss
break;

case 2  :
note = refNote+14; // D
break;

case 607  :
note  = refNote+15;  // Ess
break;

case 543  :
note = refNote+16; // E
break;

case 623  :
note = refNote+17; // F
break;

case 535  :
note = refNote+18; // Fiss
break;

case 519  :
note = refNote+19; // G
break;

case 523  :
note = refNote+20; // Giss
break;

case 515  :
note = refNote+21; // A
break;

case 635  :
note = refNote+22; // B
break;

case 539  :
note = refNote+23; // H
break;

case 537  :
note = refNote+24; // C
break;

case 621  :
note = refNote+26; // D
break;
```

```
                default:
                        note = 255;
        }
        return note;
}

void fluteMenu() {
        while ((checkFingering() & menuFing) == menuFing) { //Stannar i menyn medan knapparna för meny är nedtryckta
                PORTD |= (1 << PORTD2);
                if (checkFingering() == (menuFing + 4)){
                        PORTD &= ~(1 << PORTD2);
                        refNote++; //Knapp 4 för att transponera upp ett halvt tonsteg
                        while (checkFingering() == (menuFing + 4)); //Vänta tills knapp 4 är släppt
                } else if (checkFingering() == (menuFing + 2)) {
                        PORTD &= ~(1 << PORTD2);
                        refNote--; //Knapp 3 för att transponera ner ett halvt tonsteg
                        while (checkFingering() == (menuFing + 2)); //Vänta tills knapp 3 är släppt
                }
        }
        PORTD &= ~(1 << PORTD2);
        }

uint8_t getVelocity()
{
        uint16_t temp = 0;
        for (int i = 0; i < 4; i++)
        {
                convert_adc();
                temp += read_adc();
        }
        return (uint8_t) (temp / 4);
}

void debugNote(){
        usart0_transmit(MIDI_NOTE_ON);
        usart0_transmit(60);
        usart0_transmit(127);
        _delay_ms(1000);
        usart0_transmit(MIDI_NOTE_OFF);
        usart0_transmit(60);
        usart0_transmit(127);
        _delay_ms(1000);
}
```