Project 2 Team Report

Tim Crawford, Andrew Leise, Ben Wantland

Project description

The goal of CpE 3150 Project 2 was to create a keyboard application using the ATmega324pb microcontroller and accompanying Simon Board. This was done by utilizing the C language and Atmel Studio 7. The main program consisted of two different modes: one for playing stored songs on the microcontroller and the other for playing notes individually using the Simon Board buttons like a keyboard. The microcontroller was also connected to a host computer serially to display helpful information and to switch between modes. Upon selecting mode 1 by using the computer keyboard, the microcontroller would be able to play any stored song. There were five songs implemented and they are explained in significant detail below. During each song, the notes being played and the song title are also displayed on the terminal. Upon selecting mode 2 by using the computer keyboard, the microcontroller allows the user to play notes individually like a keyboard. Each button on the Simon Board corresponded to a specific button, with two specific switches allowing the user to play sharp notes or high notes. Again, the terminal displays the notes played in this mode as well. The portions of the project contributed by each member are explained in further detail below.

The only major problem that occurred during the implementation involved the conflict between the port bits used for the serial communication and the LED output. PORTD was utilized for outputting the note lights on the LEDs, however, a few bits of PORTD are also utilized in the serial communications. Bits 2 and 3 were both utilized in the serial communications, so they would not be suitable for LED output of any note. Bit 2 in particular, which corresponds to the receiving of characters, does not output anything because it is defined as input. Bit 3 quickly flashes from the transmissions occurring, so it also was not suited for LED output. This conflict caused confusion at times by making debugging harder and causing some reorganization of the program code. Eventually a solution was developed, and the notes that were mapped to PD2 and PD3 were remapped to other LEDs. This ensured every note had its own LED and that they all displayed properly.

Mode one allows five different songs to be played by pressing switches 1-5. The extra songs in mode one are Tim's individual contribution. The songs are played by calling a note function which handles generating the frequency to play a note, and using a timer to play the note for the required amount of time based on the bpm of the song and the length of the note. This function uses a timer with a prescale value of 1024 which lasts for about 250,000 machine cycles. A for-loop is used to loop the timer the required amount of times to play the note for the required length of time. To find how many times to loop the timer, the bpm of the song must be converted into machine cycles. This is done by taking the length of a whole note in seconds divided by the number of seconds it takes the Atmega324pb to execute an instruction. This is calculated by t_{MC} = (1cc/1MC)(1s/(16x106cc)) = 0.0000000625. It takes 2 seconds to play a whole note in 120 bpm. 2/0.0000000625 = 32,000,000 machine cycles. The timer would have to run 128 times to play the whole note. Using this calculation, the bpm of the rest of the songs can be converted. While the timer is running, the function for the current note is continuously called. These note functions call one of two timer functions that generate the proper frequency using the speaker.

A large number of notes were used in this project, so Tim wrote a python script to help with the frequency conversions. This script does the following calculation
MCs = ((1/frequency)/2)/0.0000000625 and then divides by 1024 and 256 to display timer values for both 1024 and 256 prescaling. (1/frequency)/2) is used to get the half period of the desired sound wave. 0.0000000625 is the number of seconds it takes the Atmega324pb to execute an instruction. The frequency of an A note is 220Hz. Entering this in the script results in the following values: 35.51136 for 1024 and 142.0454 for 256. One of these values would be used along with the corresponding prescaling to generate the frequency of the note through the speaker. The rest of the timer values for the notes were found using this script.

Mode two enabled the keyboard which implements all the LEDs, switches, and the buzzer. Each switch will play one of the seven notes, G - F, in order of the switches. Ben's individual contribution incorporates switches 4 and 6 which are not used for the seven notes. Switch 4, when pressed in combination with any other switch will result in the sharp version of the note being played, if the sharp exists. This will also result in the note's LED and the higher note's LED both being turned on because the sharp is in between the two notes. Switch 6, like switch 4, can also be pushed in combination with any other switch. This will result in the note being played an octave higher. In conclusion any note between G and F can be played with a combination of switches. This allows each note to be played as a normal note, a sharp, up an octave, and with both an octave just and as a sharp.

The serial communication portion of the project was implemented by Andrew. It consisted of various different functions that allowed for the ATmega324PB to communicate with an external computer. The logic corresponding to the mode selection, as well as the code for connecting and outputting the menu on the terminal was primarily Andrew's team contribution. This consisted of create multiple functions. First, the USART_Init function was created and it initializes the microcontroller ports for serial communication. Then the USART_TxChar and USART_TxString functions were implemented. These allow for characters to be transmitted to the screen terminal. See the documentation below for the program code or Andrew's individual report for more detail.

In addition to the main serial output, Andrew added the ability to change the microcontrollers mode through the keyboard interface and display the notes as they are played for his individual contribution. This consisted of adding three additional functions. The USART_RxChar function allows a character to be received. The ISR for the receiving of a character was also utilized. This allowed the program to immediately interpret the received character upon getting it in the buffer. Inside the routine, the logic for setting the new mode was implemented. Lastly, a Display_Menu function was created to display the new menu to the user on the terminal. It simply displays the menu by calling the USART_TxString function multiple times. To output the notes being played USART_TxString function is also called whenever the note is played. That concludes the main serial communications used for the project.

Outside of the external switches and LEDs on the Simon Board, a few additional external devices were utilized. A micro USB cable was connected from the Simon Board an external computer to allow for serial communication. The terminal utilized was PuTTY. PuTTY allowed the program to correctly write the entire menu and clear the screen when needed, thus it was the best choice for terminals.

See the table below for the work effort distribution.

Team Member	**Work Effort Percent**	**Contributions**
Tim Crawford	33%	Songs and Notes
Andrew Leise	33%	Serial Communication and Logic
Ben Wantland	33%	Keyboard and Logic

Work effort distribution

As stated previously, the work was split up into three different categories, so each team member did the work for one of the categories. Each part of the project was evenly distributed between the team members. Tim implemented the songs that are played when pushing switches 1-5 in mode one along with all of the notes that are used in the songs. Ben implemented the keyboard that is in mode two as well as the rest of the notes that were not used in the songs. Andrew implemented all of the serial communication.

ATmega324PB

Two of the three timers were used in this project. Timer 2 is the timer that is used in the Note function to play a note for a specific length. This timer has a prescale value of 1024 and runs for about 250,000 machine cycles. Timer 0 was used in both frequency generating functions which have 1024 and 256 as prescaler values.

Each note lights up a corresponding LED. Note G will light up LED 1 when played, A lights up LED2, B lights up LED6, C lights up LED5, D lights up LED7, E lights up LED 8, and F lights up LED 9. Sharp notes will light up the LED associated with the note as well as the note above it in the scale because the sharp falls between both notes.

In mode one, switches 1-5 can be pressed to play one of five songs that are part of the project. While in mode two, switch 1 will play a G note, 2 will play A, 3 will play B, 5 will play C, 7 will play D, 8 will play E, and 9 will play F. This mode also allows for sharps to be played when switch 4 is held. High notes can also be played in this mode when switch 6 is pressed.

For the serial communications, multiple different registers were used. The registers that were utilized consisted of the following: UBRR1L, UBRR1H, UCSR1A, UCSR1B, UCSR1C, and UDR1. The UBRR1H and UBRR1L registers were used when defining the specified baud rate of the transmissions. These were both utilized in the USART_Init function when setting up the serial communications. The desired baud rate is calculated during runtime based on a constant, BAUD_PRESCALER, that can be defined in the code. For this program, the baud rate was set to 9600, so the value passed to these registers is calculated by the following:

$$X = \frac{f}{BR*16} - 1 = \frac{16x10^6}{9600*16} - 1 = 103.167 = 103$$

Thus, the value passed to these registers is 103 and 103 shifted to the right by 8 bits (divided by 256). This sets the baud rate for the microcontroller correctly. The USCR1A register was primarily used for determining if a character has been received or checking to ensure a character has been fully transmitted. These occur in USART_RxChar and USART_TxChar respectively. The USCR1B and USCR1C registers were both used in the serial communications initialization, the USART_Init function. USCR1B was used to enable the transmitter and receiver. USCR1C was used to set the serial port for 8-bit data transfer, one stop bit, asynchronous communications. The UDR1 register was used when gathering a received value or transmitting a value. This was utilized in the USART_RxChar and USART_TxChar functions. In addition to the different registers that were utilized, the interrupt for receiving characters was also utilized. The USART_RX interrupt triggers when a character is received in the buffer and this was used to handle the menu operations upon receiving input from the keyboard.

Pin listings and functions (Simon Board)

Various components of the ATmega324pb microcontroller were utilized for the implementation of Project 2. All the bits of PORTA and PORTD were used during the different modes in the project. PORTA was used for gathering input from the switches. While all the bits of PORTD except for PORTD2 were used for LEDs and serial communication. PORTD2 was used for serial input to the board. Bits 4-6 of PORTE were also used for the speaker, LED5, and switch 5 respectively.

Pin	Device	Function
PD0	LED1	Keyboard/Songs
PD1	LED2	Keyboard/Songs
PD2	LED3	Input for Receiving Character - RXD1
PD3	LED4	Output for Transmitting Character - TXD1
PE5	LED5	Keyboard/Songs
PD4	LED6	Keyboard/Songs
PD5	LED7	Keyboard/Songs
PD6	LED8	Keyboard/Songs
PD7	LED9	Keyboard/Songs
PA0	SW1	Keyboard/Songs
PA1	SW2	Keyboard/Songs
PA2	SW3	Keyboard/Songs
PA3	SW4	Keyboard/Songs
PE6	SW5	Keyboard/Songs
PA4	SW6	Keyboard
PA5	SW7	Keyboard
PA6	SW8	Keyboard
PA7	SW9	Keyboard
PE4	Speaker	Keyboard/Songs

Table 1: ATmega324pb pin descriptions and functions

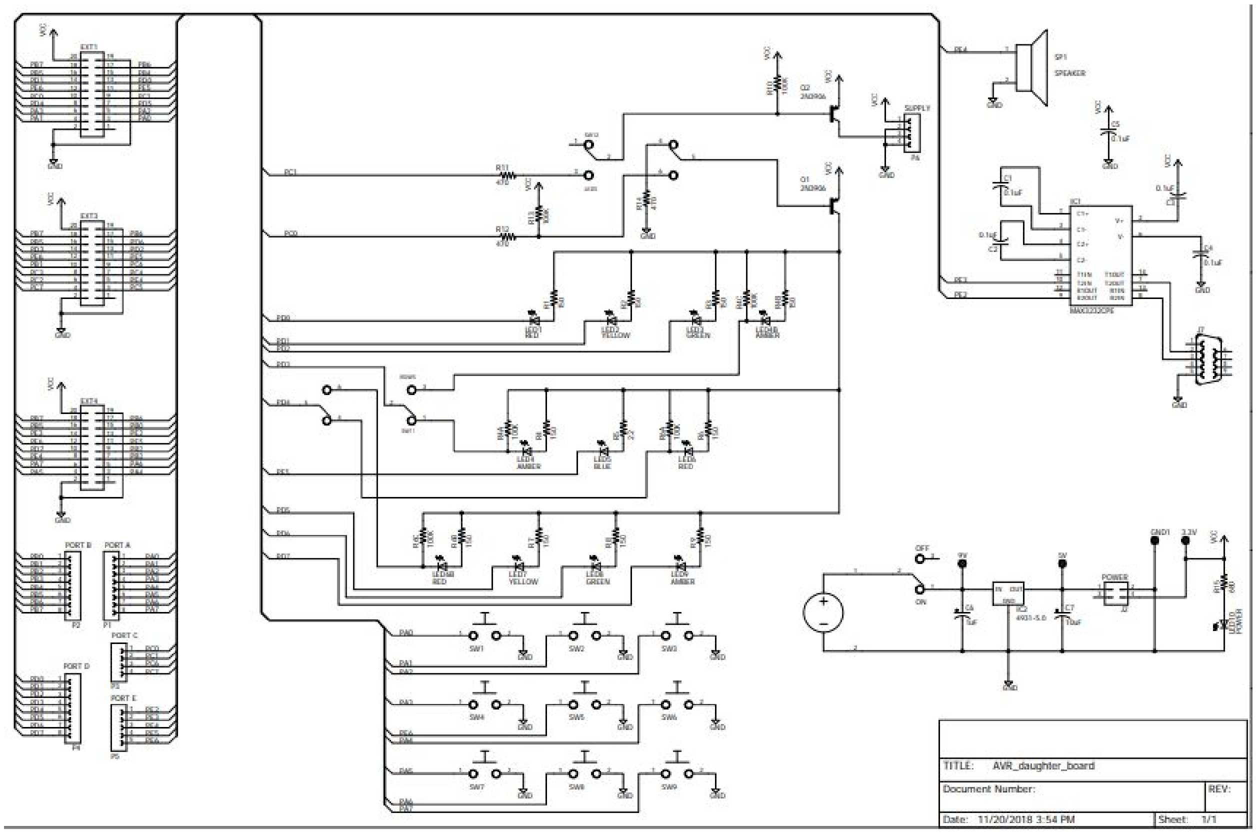

Figure 1: ATmega324pb Schematic

Commented code

```c
/*
 * Project2.c
 *
 * Created: 11/19/2019 8:37:31 PM
 * Authors : Andrew Leise, Tim Crawford, Ben Wantland
 */
// Include header files, set CPU frequency
#include <avr/io.h>
#include <avr/interrupt.h>
#include <string.h>
#define F_CPU 16000000UL
#include "util/delay.h"

// Function definitions for all functions
void USART_Init(unsigned long BAUDRATE);
char USART_RxChar();
void USART_TxChar(char data);
void USART_TxString(char*);
void Display_Menu(unsigned char selected_mode, char* song);
void Keyboard();
void Note(int bpm, float noteLength, void (*NoteToPlay)(), char* note);
void NoteTimer(int wait);
void NoteTimer256(int wait);
void Rest();
void NoteLowASharp();
void NoteLowB();
void NoteLowC();
void NoteLowD();
void NoteLowDSharp();
void NoteLowE();
void NoteLowF();
void NoteLowFSharp();
void NoteG();
void NoteA();
void NoteASharp();
void NoteB();
void NoteC();
void NoteCSharp();
void NoteD();
void NoteDSharp();
void NoteE();
void NoteF();
void NoteFSharp();
void NoteHighG();
void NoteHighGSharp();
void NoteHighA();
void NoteHighB();
void NoteHighC();
void NoteHighD();
void NoteGSharp();
void NoteHighASharp();
void NoteHighCSharp();
void NoteHighDSharp();
void NoteHighE();
void NoteHighF();
void NoteHighFSharp();
void LEDG();
void LEDA();
void LEDB();
void LEDC();
void LEDD();
void LEDE();
void LEDF();

// Include ISR for USART interface
ISR(USART1_RX_vect);

// USART1 uses PD2 for RxD1 and PD3 for TxD1

unsigned char mode = 0;

int main(void)
{
		// Define all bits except PD2 as output, PD2 as input for serial communication
		DDRD = 0xFB;

		// Ensure all LEDs are off, activate PD2's pullup resistor
		PORTD = 0xFF;

		// Define speaker and LED5 as output, SW5 as input, then pullup resistors
		DDRE = 0b00110000;
		PORTE = 0b01100000;

		// Define switches as input and activate pullup resistors
		DDRA = 0x00;
		PORTA = 0xFF;

		// Set up initial timer values and prescalers
		TCCR0A = 0x00;
		TCCR0B = 0b00000101;
		TCCR2A = 0x00;
		TCCR2B = 0b00000111;

		// Initialize USART interface with 9600 baud rate
		USART_Init(9600);

		// Enable receiving complete interrupt bit, enable interrupts
		UCSR1B |= (1 << RXCIE);
		sei();
```

```
// Display initial menu to serial output
Display_Menu(0, "");

// Enter infinite while loop
while (1)
{
	// If the current mode is song mode, check for switch to play given song
	if(mode == 0)
	{
		//Cave Story Main Theme
		if(~PINA & 0b00000001)
		{
			int bpm = 128; //Note: not actual bpm of song (120bpm)
			Display_Menu(mode, "Cave Story Main Theme");

			//Measure 1
			Note(bpm, 0.25, NoteDSharp, "D#, ");
			Note(bpm, 0.25, NoteHighGSharp, "HG#, ");
			Note(bpm, 0.25, NoteDSharp, "D#, ");
			Note(bpm, 0.25, NoteHighGSharp, "HG#, ");

			//Measure 2
			Note(bpm, 0.25, NoteD, "D, ");
			Note(bpm, 0.25, NoteHighGSharp, "HG#, ");
			Note(bpm, 0.25, NoteD, "D, ");
			Note(bpm, 0.25, NoteHighGSharp, "HG#, ");

			//Measure 3
			Note(bpm, 0.25, NoteCSharp, "C#, ");
			Note(bpm, 0.25, NoteHighGSharp, "HG#, ");
			Note(bpm, 0.25, NoteCSharp, "C#, ");
			Note(bpm, 0.25, NoteHighGSharp, "HG#, ");

			//Measure 4
			Note(bpm, 0.25, NoteC, "C, ");
			Note(bpm, 0.25, NoteHighGSharp, "HG#, ");
			Note(bpm, 0.25, NoteC, "C, ");
			Note(bpm, 0.125, NoteCSharp, "C#, ");
			Note(bpm, 0.125, NoteD, "D, ");
		}

		//Mimiga Town
		if(~PINA & 0b00000010)
		{
			int bpm = 140; //110 bpm
			Display_Menu(mode, "Mimiga Town");

			//Measure 1
			Note(bpm, 0.25, NoteLowF, "LF, ");
		Note(bpm, 0.25, NoteA, "A, ");
		Note(bpm, 0.125, NoteG, "G, ");
		Note(bpm, 0.25, NoteLowF, "LF, ");
		Note(bpm, 0.125, NoteLowF, "LF, ");

		//Measure 2
		Note(bpm, 0.25, NoteC, "C, ");
		Note(bpm, 0.125, NoteA, "A, ");
		Note(bpm, 0.375, NoteC, "C, ");
		Note(bpm, 0.25, NoteC, "C, ");

		//Measure 3
		Note(bpm, 0.25, NoteD, "D, ");
		Note(bpm, 0.25, NoteC, "C, ");
		Note(bpm, 0.25, NoteF, "F, ");
		Note(bpm, 0.25, NoteC, "C, ");

		//Measure 4
		Note(bpm, 0.75, NoteASharp, "A#, ");
	}

	//On to Grasstown
	if(~PINA & 0b00000100)
	{
		int bpm = 96; //160 bpm
		Display_Menu(mode, "Grasstown");

		//Measure 1
		Note(bpm, 0.0625, NoteG, "G, ");
		Note(bpm, 0.0625, Rest, "");
		Note(bpm, 0.125, NoteLowC, "LC, ");
		Note(bpm, 0.125, Rest, "");
		Note(bpm, 0.125, NoteG, "G, ");
		Note(bpm, 0.0625, NoteA, "A, ");
		Note(bpm, 0.0625, Rest, "");
		Note(bpm, 0.125, NoteG, "G, ");
		Note(bpm, 0.125, Rest, "");
		Note(bpm, 0.125, NoteLowC, "LC, ");

		//Measure 2
		Note(bpm, 0.1875, NoteG, "G, ");
		Note(bpm, 0.1875, NoteLowF, "LF, ");
		Note(bpm, 0.125, NoteLowE, "LE, ");
		Note(bpm, 0.125, Rest, "");
		Note(bpm, 0.125, NoteLowE, "LE, ");
		Note(bpm, 0.125, NoteLowD, "LD, ");
		Note(bpm, 0.125, NoteLowE, "LE, ");

		//Measure 3
		Note(bpm, 0.375, NoteLowF, "LF, ");
		Note(bpm, 0.0625, NoteLowF, "LF, ");
		Note(bpm, 0.0625, NoteLowE, "LE, ");
		Note(bpm, 0.375, NoteLowD, "LD, ");
		Note(bpm, 0.0625, NoteLowF, "LF, ");
		Note(bpm, 0.0625, NoteLowE, "LE, ");

		//Measure 4
		Note(bpm, 1, NoteLowD, "LD, ");
}

//Meltdown 2
if(~PINA & 0b00001000)
{
		int bpm = 110; //140 bpm
		Display_Menu(mode, "Meltdown 2");

		//Measure 21
		Note(bpm, 0.1875, NoteHighG, "HG, ");
		Note(bpm, 0.0625, NoteFSharp, "F#, ");
		Note(bpm, 0.1875, NoteHighG, "HG, ");
		Note(bpm, 0.0625, NoteFSharp, "F#, ");
		Note(bpm, 0.125, NoteHighG, "HG, ");
		Note(bpm, 0.0625, NoteE, "E, ");
		Note(bpm, 0.0625, NoteFSharp, "F#, ");
		Note(bpm, 0.125, NoteHighG, "HG, ");
		Note(bpm, 0.125, NoteHighA, "HA, ");

		//Measure 22
		Note(bpm, 0.125, NoteHighB, "HB, ");
		Note(bpm, 0.125, NoteHighC, "HC, ");
		Note(bpm, 0.375, NoteHighD, "HD, ");
		Note(bpm, 0.0625, NoteHighC, "HC, ");
		Note(bpm, 0.0625, NoteHighB, "HB, ");
		Note(bpm, 0.125, NoteHighA, "HA, ");
		Note(bpm, 0.125, NoteFSharp, "F#, ");

		//Measure 23
		Note(bpm, 0.1875, NoteHighA, "HA, ");
		Note(bpm, 0.0625, NoteHighG, "HG, ");
		Note(bpm, 0.1875, NoteHighA, "HA, ");
		Note(bpm, 0.0625, NoteHighG, "HG, ");
		Note(bpm, 0.125, NoteFSharp, "F#, ");
		Note(bpm, 0.125, NoteE, "E, ");
		Note(bpm, 0.125, NoteD, "D, ");
		Note(bpm, 0.125, NoteE, "E, ");

		//Measure 24
		Note(bpm, 0.0625, NoteD, "D, ");
		Note(bpm, 0.0625, NoteE, "E, ");
		Note(bpm, 0.875, NoteD, "D, ");
```

```
}

//Balrog's Theme
if (~PINE & 0b01000000)
{
        int bpm = 106; //145 bpm
        Display_Menu(mode, "Balrog's Theme");

        //Measure 1
        Note(bpm, 0.0625, NoteLowC, "LC, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowC, "LC, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowFSharp, "LF#, ");
        Note(bpm, 0.1875, Rest, "");
        Note(bpm, 0.0625, NoteLowF, "LF, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowF, "LF, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowFSharp, "LF#, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteG, "G, ");
        Note(bpm, 0.0625, Rest, "");

        //Measure 2
        Note(bpm, 0.0625, NoteG, "G, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteG, "G, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowFSharp, "LF#, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowF, "LF, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowDSharp, "LD#, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowDSharp, "LD#, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.1875, NoteLowC, "LC, ");
        Note(bpm, 0.0625, NoteLowB, "LB, ");

        //Measure 3
        Note(bpm, 0.0625, NoteLowASharp, "LA#, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowASharp, "LA#, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowF, "LF, ");
        Note(bpm, 0.1875, Rest, "");
        Note(bpm, 0.0625, NoteLowE, "LE, ");
        Note(bpm, 0.0625, Rest, "");
        Note(bpm, 0.0625, NoteLowE, "LE, ");
```

```
				Note(bpm, 0.0625, Rest, "");
				Note(bpm, 0.0625, NoteLowF, "LF, ");
				Note(bpm, 0.0625, Rest, "");
				Note(bpm, 0.0625, NoteLowFSharp, "LF#, ");
				Note(bpm, 0.0625, Rest, "");

				//Measure 4
				Note(bpm, 0.0625, NoteG, "G, ");
				Note(bpm, 0.0625, Rest, "");
				Note(bpm, 0.0625, NoteLowFSharp, "LF#, ");
				Note(bpm, 0.0625, Rest, "");
				Note(bpm, 0.0625, NoteLowF, "LF, ");
				Note(bpm, 0.0625, Rest, "");
				Note(bpm, 0.0625, NoteLowF, "LF, ");
				Note(bpm, 0.0625, Rest, "");
				Note(bpm, 0.0625, NoteLowDSharp, "LD#, ");
				Note(bpm, 0.0625, Rest, "");
				Note(bpm, 0.0625, NoteLowDSharp, "LD#, ");
				Note(bpm, 0.0625, Rest, "");
				Note(bpm, 0.125, NoteLowC, "LC, ");
				Note(bpm, 0.0625, Rest, "");
			}
		}
		// If mode is keyboard mode, play specific note for switch
		if(mode == 1)
		{
			if((~PINA & 0x01) && !(~PINA & 0x08) && !(~PINA & 0x10)) //normal
			{
				USART_TxString("G, ");
				while((~PINA & 0x01) && !(~PINA & 0x08) && !(~PINA & 0x10))
				{
					NoteG(); //G note
				}
			}
			if((~PINA & 0x01) && (~PINA & 0x08) && !(~PINA & 0x10)) //sharp
			{
				USART_TxString("G#, ");
				while((~PINA & 0x01) && (~PINA & 0x08) && !(~PINA & 0x10))
				{
					NoteGSharp();//G# note
				}
			}
			if((~PINA & 0x01) && !(~PINA & 0x08) && (~PINA & 0x10)) //octave
			{
				USART_TxString("HG, ");
				while((~PINA & 0x01) && !(~PINA & 0x08) && (~PINA & 0x10))
				{
					NoteHighG();//High G note
				}
				}
```

```
if((~PINA & 0x01) && (~PINA & 0x08) && (~PINA & 0x10)) //sharp octave
{
		USART_TxString("HG#, ");
		while((~PINA & 0x01) && (~PINA & 0x08) && (~PINA & 0x10))
		{
				NoteHighGSharp();//High G# note
		}
}
if((~PINA & 0x02) && !(~PINA & 0x08) && !(~PINA & 0x10)) //normal
{
		USART_TxString("A, ");
		while((~PINA & 0x02) && !(~PINA & 0x08) && !(~PINA & 0x10))
		{
				NoteA();//A note
		}
}
if((~PINA & 0x02) && (~PINA & 0x08) && !(~PINA & 0x10)) //sharp
{
		USART_TxString("A#, ");
		while((~PINA & 0x02) && (~PINA & 0x08) && !(~PINA & 0x10))
		{
				NoteASharp();//A# note
		}
}
if((~PINA & 0x02) && !(~PINA & 0x08) && (~PINA & 0x10)) //octave
{
		USART_TxString("HA, ");
		while((~PINA & 0x02) && !(~PINA & 0x08) && (~PINA & 0x10))
		{
				NoteHighA();//High A note
		}
}
if((~PINA & 0x02) && (~PINA & 0x08) && (~PINA & 0x10)) //sharp octave
{
		USART_TxString("HA#, ");
		while((~PINA & 0x02) && (~PINA & 0x08) && (~PINA & 0x10))
		{
				NoteHighASharp();//High A# note
		}
}
if((~PINA & 0x04) && !(~PINA & 0x08) && !(~PINA & 0x10)) //normal
{
		USART_TxString("B, ");
		while((~PINA & 0x04) && !(~PINA & 0x08) && !(~PINA & 0x10))
		{
				NoteB();//B note
		}
}
if((~PINA & 0x04) && (~PINA & 0x08) && !(~PINA & 0x10)) //sharp
{
		USART_TxString("B, ");
		while((~PINA & 0x04) && (~PINA & 0x08) && !(~PINA & 0x10))
		{
			NoteB();//B note
		}
	}
	if((~PINA & 0x04) && !(~PINA & 0x08) && (~PINA & 0x10)) //octave
	{
		USART_TxString("HB, ");
		while((~PINA & 0x04) && !(~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighB();//High B note
		}
	}
	if((~PINA & 0x04) && (~PINA & 0x08) && (~PINA & 0x10)) //sharp octave
	{
		USART_TxString("HB, ");
		while((~PINA & 0x04) && (~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighB();//High B note
		}
	}
	if((~PINE & 0x40) && !(~PINA & 0x08) && !(~PINA & 0x10)) //normal
	{
		USART_TxString("C, ");
		while((~PINE & 0x40) && !(~PINA & 0x08) && !(~PINA & 0x10))
		{
			NoteC();//C note
		}
	}
	if((~PINE & 0x40) && (~PINA & 0x08) && !(~PINA & 0x10)) //sharp
	{
		USART_TxString("C#, ");
		while((~PINE & 0x40) && (~PINA & 0x08) && !(~PINA & 0x10))
		{
			NoteCSharp();//C# note
		}
	}
	if((~PINE & 0x40) && !(~PINA & 0x08) && (~PINA & 0x10)) //octave
	{
		USART_TxString("HC, ");
		while((~PINE & 0x40) && !(~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighC();//High C note
		}
	}
	if((~PINE & 0x40) && (~PINA & 0x08) && (~PINA & 0x10)) //sharp octave
	{
		USART_TxString("HC#, ");
		while((~PINE & 0x40) && (~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighCSharp();//High C# note
		}
	}
	if((~PINA & 0x20) && !(~PINA & 0x08) && !(~PINA & 0x10)) //normal
	{
		USART_TxString("D, ");
		while((~PINA & 0x20) && !(~PINA & 0x08) && !(~PINA & 0x10))
		{
			NoteD();//D note
		}
	}
	if((~PINA & 0x20) && (~PINA & 0x08) && !(~PINA & 0x10)) //sharp
	{
		USART_TxString("D#, ");
		while((~PINA & 0x20) && (~PINA & 0x08) && !(~PINA & 0x10))
		{
			NoteDSharp(); //D# note
		}
	}
	if((~PINA & 0x20) && !(~PINA & 0x08) && (~PINA & 0x10)) //octave
	{
		USART_TxString("HD, ");
		while((~PINA & 0x20) && !(~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighD();//High D note
		}
	}
	if((~PINA & 0x20) && (~PINA & 0x08) && (~PINA & 0x10)) //sharp octave
	{
		USART_TxString("HD#, ");
		while((~PINA & 0x20) && (~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighDSharp();//High D# note
		}
	}
	if((~PINA & 0x40) && !(~PINA & 0x08) && !(~PINA & 0x10)) //normal
	{
		USART_TxString("E, ");
		while((~PINA & 0x40) && !(~PINA & 0x08) && !(~PINA & 0x10))
		{
			NoteE();//E note
		}
	}
	if((~PINA & 0x40) && (~PINA & 0x08) && !(~PINA & 0x10)) //sharp
	{
		USART_TxString("E, ");
		while((~PINA & 0x40) && (~PINA & 0x08) && !(~PINA & 0x10))
		{
			NoteE();//E note
		}
	}
	if((~PINA & 0x40) && !(~PINA & 0x08) && (~PINA & 0x10)) //octave
	{
		USART_TxString("HE, ");
		while((~PINA & 0x40) && !(~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighE();//High E note
		}
	}
	if((~PINA & 0x40) && (~PINA & 0x08) && (~PINA & 0x10)) //sharp octave
	{
		USART_TxString("HE, ");
		while((~PINA & 0x40) && (~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighE();//High E note
		}
	}
	if((~PINA & 0x80) && !(~PINA & 0x08) && !(~PINA & 0x10)) //normal
	{
		USART_TxString("F, ");
		while((~PINA & 0x80) && !(~PINA & 0x08) && !(~PINA & 0x10))
		{
			NoteF();//F note
		}
	}
	if((~PINA & 0x80) && (~PINA & 0x08) && !(~PINA & 0x10)) //sharp
	{
		USART_TxString("F#, ");
		while((~PINA & 0x80) && (~PINA & 0x08) && !(~PINA & 0x10))
		{
			NoteFSharp();//F# note
		}
	}
	if((~PINA & 0x80) && !(~PINA & 0x08) && (~PINA & 0x10)) //octave
	{
		USART_TxString("HF, ");
		while((~PINA & 0x80) && !(~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighF();//High F note
		}
	}
	if((~PINA & 0x80) && (~PINA & 0x08) && (~PINA & 0x10)) //sharp octave
	{
		USART_TxString("HF#, ");
		while((~PINA & 0x80) && (~PINA & 0x08) && (~PINA & 0x10))
		{
			NoteHighFSharp();//High F# note
		}
	}
```

```
                }
        }
}

// Function implmemtation for USART_Init
void USART_Init(unsigned long BAUDRATE)
{
        // Calculate baud prescaler value and set low/high bytes of baud rate register
        int BAUD_PRESCALER = (F_CPU / (BAUDRATE * 16UL)) - 1;
        UBRR1L = BAUD_PRESCALER;
        UBRR1H = (BAUD_PRESCALER >> 8);

        // Enable transmitter and receiver in status control register B
        UCSR1B |= (1 << RXEN) | (1 << TXEN);

        // Set bits corresponding to 8-bit data transfer, one stop bit, asychronous
        UCSR1C |= (1 << UCSZ0) | (1 << UCSZ1);
}

// Function implementation for UART_RxChar
char USART_RxChar()
{
        // Check to see if character has been received
        if(UCSR1A & (1 << RXC))
        {
                // If char is received, return it
                return UDR1;
        }
        else
        {
                // If no char is received, return null
                return '\0';
        }
}

// Function implementation for USART_TxChar
void USART_TxChar(char data)
{
        // Set data to UDR1 register, then loop while data is being transferred
        UDR1 = data;
        while(!(UCSR1A & (1 << UDRE)));
}

// Function implementation for USART_TxString
void USART_TxString(char *str)
{
        // Loop while characters still exist in the string, and transmit each character individually
        while(*str != '\0')
        {
                USART_TxChar(*str++);
```

```
		}
	}

	// Interrupt service routine implementation
	ISR(USART1_RX_vect)
	{
		// Set character to received value
		unsigned char val = UDR1;

		// If a 1 was received, change to mode 0 (if not already in mode 0) and output new menu
		if(val == '1')
		{
			if(mode != 0)
			{
				mode = 0;
				Display_Menu(mode, "");
			}
		}
		// If a 2 was received, change to mode 1 (if not already in mode 1) and output new menu
		else if(val == '2')
		{
			if(mode != 1)
			{
				mode = 1;
				Display_Menu(mode, "");
			}
		}
	}

	// Function implementation for Display_Menu
	void Display_Menu(unsigned char selected_mode, char* song)
	{
		// Clear terminal by calling PuTTY specific command
		USART_TxChar(27);
		USART_TxString("[2J");
		USART_TxChar(27);
		USART_TxString("[H");

		// Output new menu to the terminal
		USART_TxString("Song selection menu\r\n");
		USART_TxString("Select one of the following modes!\r\n");
		USART_TxString("1 - Song Mode: Play one of the stored songs\r\n");
		USART_TxString("2 - Keyboard Mode: Play your own tunes!\r\n");

		// Display the current mode and song based on passed values
		// Song mode
		if (selected_mode == 0)
		{
			// No song is passed
			if (song[0] == '\0')
```

```
				{
					USART_TxString("Current Mode = 1\r\n\n");
				}
				// Song is being played
				else
				{
					USART_TxString("Current Mode = 1\r\n\n");
					USART_TxString("Current Song = ");
					USART_TxString(song);
					USART_TxString("\r\n\n");
					USART_TxString("Played Keys:\r\n");
				}
		}
		// Keyboard mode
		else if(selected_mode == 1)
		{
				USART_TxString("Current Mode = 2\r\n\n");
				USART_TxString("Played Keys:\r\n");
		}
}

// Play a specified note for the length of the note
void Note(int bpm, float noteLength, void (*NoteToPlay)(), char* note)
{
		USART_TxString(note);
		for(int i = 0; i < bpm*noteLength; i++)
		{
				TCNT2 = -244;
				while (!(TIFR2 & (1<<TOV2)))
				{
					NoteToPlay();
				}
				TIFR2 = 1<<TOV2;
		}
}

// Timer with 1024 prescaling
void NoteTimer(int wait)
{
		TCNT0 = wait;
		TCCR0A = 0x00;
		TCCR0B = 0b00000101;
		while (!(TIFR0 & (1<<TOV0)));
		PORTE ^= 0b00010000;
		TIFR0 = 1<<TOV0;
}

// Timer with 256 prescaling
void NoteTimer256(int wait)
{
		TCNT0 = wait;
		TCCR0A = 0x00;
		TCCR0B = 0b00000100;
		while (!(TIFR0 & (1<<TOV0)));
		PORTE ^= 0b00010000;
		TIFR0 = 1<<TOV0;
}

// The most important function in the entire project
void Rest(){}

// All of the note functions are after this point
void NoteLowASharp()
{
		LEDA();
		LEDB();
		NoteTimer(-67);
		LEDA();
		LEDB();
}

void NoteLowB()
{
		LEDB();
		NoteTimer256(-253);
		LEDB();
}

void NoteLowC()
{
		LEDC();
		NoteTimer256(-239);
		LEDC();
}

void NoteLowD()
{
		LEDD();
		NoteTimer(-53);
		LEDD();
}

void NoteLowDSharp()
{
		LEDD();
		LEDE();
		NoteTimer256(-201);
		LEDD();
		LEDE();
}
```

```
void NoteLowE()
{
		LEDE();
		NoteTimer(-47);
		LEDE();
}

void NoteLowF()
{
		LEDF();
		NoteTimer(-45);
		LEDF();
}

void NoteLowFSharp()
{
		LEDF();
		LEDG();
		NoteTimer256(-169);
		LEDF();
		LEDG();
}

void NoteG()
{
		LEDG();
		NoteTimer(-40);
		LEDG();
}

void NoteA()
{
		LEDA();
		NoteTimer(-35);
		LEDA();
}

void NoteASharp()
{
		LEDA();
		LEDB();
		NoteTimer256(-134);
		LEDA();
		LEDB();
}

void NoteB()
{
		LEDB();
		NoteTimer(-31);
		LEDB();
}

void NoteC()
{
		LEDC();
		NoteTimer(-30);
		LEDC();
}

void NoteCSharp()
{
		LEDC();
		LEDD();
		NoteTimer(-28);
		LEDC();
		LEDD();
}

void NoteD()
{
		LEDD();
		NoteTimer(-26);
		LEDD();
}

void NoteDSharp()
{
		LEDD();
		LEDE();
		NoteTimer(-25);
		LEDD();
		LEDE();
}

void NoteE()
{
		LEDE();
		NoteTimer(-24);
		LEDE();
}

void NoteF()
{
		LEDF();
		NoteTimer256(-89);
		LEDF();
}
```

```
void NoteFSharp()
{
		LEDF();
		LEDG();
		NoteTimer(-21);
		LEDF();
		LEDG();
}

void NoteHighG()
{
		LEDG();
		NoteTimer256(-80);
		LEDG();
}

void NoteHighGSharp()
{
		LEDG();
		LEDA();
		NoteTimer256(-75);
		LEDG();
		LEDA();
}

void NoteHighA()
{
		LEDA();
		NoteTimer256(-71);
		LEDA();
}

void NoteHighB()
{
		LEDB();
		NoteTimer256(-63);
		LEDB();
}

void NoteHighC()
{
		LEDC();
		NoteTimer256(-60);
		LEDC();
}

void NoteHighD()
{
		LEDD();
		NoteTimer256(-53);
```

```
        LEDD();
}

void NoteGSharp()
{
        LEDG();
        LEDA();
        NoteTimer(-19);
        LEDG();
        LEDA();
}

void NoteHighASharp()
{
        LEDA();
        LEDB();
        NoteTimer(-17);
        LEDA();
        LEDB();
}

void NoteHighCSharp()
{
        LEDC();
        LEDD();
        NoteTimer(-14);
        LEDC();
        LEDD();
}

void NoteHighDSharp()
{
        LEDD();
        LEDE();
        NoteTimer(-13);
        LEDD();
        LEDE();
}
void NoteHighE()
{
        LEDE();
        NoteTimer(-12);
        LEDE();
}

void NoteHighF()
{
        LEDF();
        LEDG();
        NoteTimer(-11);
```

```
		LEDF();
		LEDG();
}

void NoteHighFSharp()
{
		LEDF();
		LEDG();
		NoteTimer(-10);
		LEDF();
		LEDG();
}


//LEDs for each note
void LEDG()
{
		PORTD ^= 0b00000001;
}

void LEDA()
{
		PORTD ^= 0b00000010;
}

void LEDB()
{
		PORTD ^= 0b00010000;
}

void LEDC()
{
		PORTE ^= 0b00100000;
}

void LEDD()
{
		PORTD ^= 0b00100000;
}

void LEDE()
{
		PORTD ^= 0b01000000;
}

void LEDF()
{
		PORTD ^= 0b10000000;
}
```