Lab 9: Pulse Width Modulator (music player) (1 Day)

EE120B Section 22

Matthew Coyne
CSID: mcoyn001

Ashley McDaniel
CSID: amcda005

Part #1

```
#include <avr/io.h>
#include "timer.h"


// 0.954 hz is lowest frequency possible with this function,
// based on settings in PWM_on()
// Passing in 0 as the frequency will stop the speaker from generating sound
void set_PWM(double frequency) {
	static double current_frequency; // Keeps track of the currently set frequency
	// Will only update the registers when the frequency changes, otherwise allows
	// music to play uninterrupted.
	if (frequency != current_frequency) {
	if (!frequency) { TCCR0B &= 0x08; } //stops timer/counter
	else { TCCR0B |= 0x03; } // resumes/continues timer/counter

	// prevents OCR3A from overflowing, using prescaler 64
	// 0.954 is smallest frequency that will not result in overflow
	if (frequency < 0.954) { OCR0A = 0xFFFF; }

	// prevents OCR0A from underflowing, using prescaler 64                    // 31250 is
largest frequency that will not result in underflow
	else if (frequency > 31250) { OCR0A = 0x0000; }

	// set OCR3A based on desired frequency
	else { OCR0A = (short)(8000000 / (128 * frequency)) - 1; }

	TCNT0 = 0; // resets counter
	current_frequency = frequency; // Updates the current frequency
	}
}

void PWM_on() {
	TCCR0A = (1 << COM0A0) | (1 << WGM00);
	// COM3A0: Toggle PB3 on compare match between counter and OCR0A
	TCCR0B = (1 << WGM02) | (1 << CS01) | (1 << CS00);
	// WGM02: When counter (TCNT0) matches OCR0A, reset counter
	// CS01 & CS30: Set a prescaler of 64
	set_PWM(0);
}

void PWM_off() {
```

```
	TCCR0A = 0x00;
	TCCR0B = 0x00;
}


enum States {OFF, NOTE_C4, NOTE_D4, NOTE_E4, HOLD}state;
double c4 = 261.63;
double d4 = 293.66;
double e4 = 329.63;

void Tick()
{
	unsigned char tempA = (~PINA & 0x07);

	switch(state)
	{
	case OFF:
	if(tempA == 0x01){
		state = NOTE_C4;
	}
	else if(tempA == 0x02){
		state = NOTE_D4;
	}
	else if(tempA == 0x04){
		state = NOTE_E4;
	}
	else{
		state = OFF;
	}
	break;
	case NOTE_C4:
	if(tempA == 0x01){
		state = NOTE_C4;
	}else{
		state = OFF;
	}
	break;
	case NOTE_D4:
	if(tempA == 0x02){
	state = NOTE_D4;
	}else{
	state = OFF;
	}
```

```
		break;
		case NOTE_E4:
		if(tempA == 0x04){
			state = NOTE_E4;
		}else{
			state = OFF;
		}
		break;
		default:
		state = OFF;
		break;
		}

		switch(state)
		{
		case OFF:
		set_PWM(0);
		break;
		case NOTE_C4:
		set_PWM(c4);
		break;
		case NOTE_D4:
		set_PWM(d4);
		break;
		case NOTE_E4:
		set_PWM(e4);
		break;
		case HOLD:
		set_PWM(0);
		break;
		default:
		set_PWM(0);
		break;
		}
}

int main()
{
		DDRA = 0x00; PORTA = 0xFF;
		DDRB = 0xFF; PORTB = 0x00;
		DDRD = 0xFF; PORTD = 0x00;
```

```
	PWM_on();
	set_PWM(0);

	state = OFF;
	TimerFlag = 0;
	//unsigned char tempA;
	while (1)
	{
	Tick();
//	tempA = ~PINA; // Input is reversed to be activated on low, so we need to bitwise inverse
PINA to get the same kind of logic as usual.
//	if( (tempA & 0x01) == 0x01){
//	set_PWM(440.00);
//	PORTD = 0xFF;
//	}
//	while(!TimerFlag);
//	TimerFlag = 0;

	}
	return 0;
}
```

Part #2

```
#include <avr/io.h>
#include "timer.h"


// 0.954 hz is lowest frequency possible with this function,
// based on settings in PWM_on()
// Passing in 0 as the frequency will stop the speaker from generating sound
void set_PWM(double frequency) {
	static double current_frequency; // Keeps track of the currently set frequency
	// Will only update the registers when the frequency changes, otherwise allows
	// music to play uninterrupted.
	if (frequency != current_frequency) {
	if (!frequency) { TCCR0B &= 0x08; } //stops timer/counter
	else { TCCR0B |= 0x03; } // resumes/continues timer/counter

	// prevents OCR3A from overflowing, using prescaler 64
	// 0.954 is smallest frequency that will not result in overflow
	if (frequency < 0.954) { OCR0A = 0xFFFF; }
```

```
	// prevents OCR0A from underflowing, using prescaler 64		// 31250 is
largest frequency that will not result in underflow
	else if (frequency > 31250) { OCR0A = 0x0000; }

	// set OCR3A based on desired frequency
	else { OCR0A = (short)(8000000 / (128 * frequency)) - 1; }

	TCNT0 = 0; // resets counter
	current_frequency = frequency; // Updates the current frequency
	}
}

void PWM_on() {
	TCCR0A = (1 << COM0A0) | (1 << WGM00);
	// COM3A0: Toggle PB3 on compare match between counter and OCR0A
	TCCR0B = (1 << WGM02) | (1 << CS01) | (1 << CS00);
	// WGM02: When counter (TCNT0) matches OCR0A, reset counter
	// CS01 & CS30: Set a prescaler of 64
	set_PWM(0);
}

void PWM_off() {
	TCCR0A = 0x00;
	TCCR0B = 0x00;
}


enum States {OFF, INCREASE, DECREASE} state;
unsigned short system_period = 80; // 1/16th of a second is 62.5 ms, just round to 63;
//unsigned char bpm = 120; // 2 beats per second
//unsigned char bpm = 60; // Let's start with 1 beat per second to make it easy and adjust later
maybe.
unsigned char eigth = 2 ; // ticks per note
unsigned char quarter = 6; // ticks per note
unsigned char half = 8; // ticks per note
unsigned char whole = 16;

double c4 = 261.63;
double d4 = 293.66;
double e4 = 329.63;
double f4 = 349.23;
double g4 = 392.00;
double a4 = 440.00;
```

```
double b4 = 493.88;
double c5 = 523.25;
double d5 = 587.33;
double e5 = 659.25;
double f5 = 698.46;
double g5 = 783.99;

unsigned char notes_number = 20;
unsigned char position = 0, count = 0;
void notes(){
   double music_notes[8] = {c4,d4,e4,f4,g4,a4,b4,c5};
   unsigned char note_length[] = {eigth,1,eigth,1,eigth,half,half,eigth,eigth,eigth, half, quarter,
eigth,eigth,eigth, half, quarter, eigth,eigth,eigth,whole};

   unsigned char tempA = (~PINA & 0x01);
   switch(state){
         case OFF:
               if(tempA == 0x02)
               {
                     if(7 > count)
                     {
                           count ++;
                     }
                     set_PWM(music_notes[count]);
                     state = INCREASE;
               }
               else if(tempA == 0x04)
               {
                     if(count != 0)
                     {
                           count --;
                     }
                     set_PWM(music_notes[count]);
                     state = DECREASE;
               }
               else
               {
                     state = OFF;
               }
               break;
         case INCREASE:
               if(tempA == 0x02)
               {
```

```
                        state = INCREASE;
                }
                else
                {
                        state = OFF;
                }
                break;
        default:
                state = OFF;
                break;
   }
}
char isItPlaying = 0;
enum playing {press, PLAY} playState;
void Tick()
{

        double music_notes[8] = {c4,d4,e4,f4,g4,a4,b4,c5};
        unsigned char note_length[] = {eigth,1,eigth,1,eigth,half,half,eigth,eigth,eigth, half,
quarter, eigth,eigth,eigth, half, quarter, eigth,eigth,eigth,whole};
   count = 0;
        unsigned char tempA = (~PINA & 0x01);

        switch(playState)
        {
        case PLAY:
        if(tempA == 0x01)
        {
                if(!PLAY)
                {
                        playState = press;
                        PWM_on();
                        isItPlaying = 1;
                        set_PWM(music_notes[count]);
                }
                else
                {
                        PWM_off();
                        isItPlaying = 0;
                        playState = press;
                }
        }
        else
```

```
        {
                playState = PLAY;
        }
        break;

        case press:
        if(tempA == 0x01)
        {
                playState = press;
        }
        else
        {
        playState = PLAY;
        }
        break;
        default:
        playState = PLAY;
        break;
   }


}

int main()
{
        DDRA = 0x00; PORTA = 0xFF;
        DDRB = 0xFF; PORTB = 0x00;
        DDRD = 0xFF; PORTD = 0x00;


        PWM_on();
        set_PWM(0);

        TimerSet(system_period);
        TimerOn();

        state = OFF;
        TimerFlag = 0;
        count = 0;
        position = 0;
        while (1)
        {
         notes();
```

```
	Tick();
	// Input is reversed to be activated on low, so we need to bitwise inverse PINA to get the
same kind of logic as usual.
	while(!TimerFlag);
	TimerFlag = 0;
	}
	return 0;
}
```

Part 3:

```
#include <avr/io.h>
#include "timer.h"


// 0.954 hz is lowest frequency possible with this function,
// based on settings in PWM_on()
// Passing in 0 as the frequency will stop the speaker from generating sound
void set_PWM(double frequency) {
	static double current_frequency; // Keeps track of the currently set frequency
	// Will only update the registers when the frequency changes, otherwise allows
	// music to play uninterrupted.
	if (frequency != current_frequency) {
	if (!frequency) { TCCR0B &= 0x08; } //stops timer/counter
	else { TCCR0B |= 0x03; } // resumes/continues timer/counter

	// prevents OCR3A from overflowing, using prescaler 64
	// 0.954 is smallest frequency that will not result in overflow
	if (frequency < 0.954) { OCR0A = 0xFFFF; }

	// prevents OCR0A from underflowing, using prescaler 64                    // 31250 is
largest frequency that will not result in underflow
	else if (frequency > 31250) { OCR0A = 0x0000; }

	// set OCR3A based on desired frequency
	else { OCR0A = (short)(8000000 / (128 * frequency)) - 1; }

	TCNT0 = 0; // resets counter
	current_frequency = frequency; // Updates the current frequency
	}
}

void PWM_on() {
	TCCR0A = (1 << COM0A0) | (1 << WGM00);
```

```
	// COM3A0: Toggle PB3 on compare match between counter and OCR0A
	TCCR0B = (1 << WGM02) | (1 << CS01) | (1 << CS00);
	// WGM02: When counter (TCNT0) matches OCR0A, reset counter
	// CS01 & CS30: Set a prescaler of 64
	set_PWM(0);
}

void PWM_off() {
	TCCR0A = 0x00;
	TCCR0B = 0x00;
}


enum States {OFF, PLAY} state;
unsigned short system_period = 80; // 1/16th of a second is 62.5 ms, just round to 63;
//unsigned char bpm = 120; // 2 beats per second
//unsigned char bpm = 60; // Let's start with 1 beat per second to make it easy and adjust later
maybe.
unsigned char eigth = 2 ; // ticks per note
unsigned char quarter = 6; // ticks per note
unsigned char half = 8; // ticks per note
unsigned char whole = 16;

double c4 = 261.63;
double d4 = 293.66;
double e4 = 329.63;
double f4 = 349.23;
double g4 = 392.00;
double a4 = 440.00;
double b4 = 493.88;
double c5 = 523.25;
double d5 = 587.33;
double e5 = 659.25;
double f5 = 698.46;
double g5 = 783.99;

unsigned char notes_number = 20;
unsigned char position = 0, count = 0;

void Tick()
{
	double music_notes[] =
{d4,0.0,d4,0.0,d4,g4,d5,c5,b4,a4,g5,d5,c5,b4,a4,g5,d5,c5,b4,c5,a4};
```

```
	unsigned char note_length[] = {eigth,1,eigth,1,eigth,half,half,eigth,eigth,eigth, half,
quarter, eigth,eigth,eigth, half, quarter, eigth,eigth,eigth,whole};

	unsigned char tempA = (~PINA & 0x01);

	switch(state)
	{
	case OFF:
	if(tempA == 0x01){
	state = PLAY;
	}
	else{
	state = OFF;
	}
	break;
	case PLAY:
	if( (position >=notes_number) && (count >= note_length[position])){
	state = OFF;
	}
	break;
	default:
	state = OFF;
	break;
	}

	switch(state)
	{
	case OFF:
	set_PWM(0);
	count = 0;
	position = 0;
	break;
	case PLAY:
	if(count < note_length[position]){
		set_PWM(music_notes[position]);
	}else {
		position++;
		count = 0;
		set_PWM(music_notes[position]);
	}
	count++;
	break;
	default:
```

```
        set_PWM(0);
        break;
        }
}

int main()
{
        DDRA = 0x00; PORTA = 0xFF;
        DDRB = 0xFF; PORTB = 0x00;
        DDRD = 0xFF; PORTD = 0x00;


        PWM_on();
        set_PWM(0);

        TimerSet(system_period);
        TimerOn();

        state = OFF;
        TimerFlag = 0;
        count = 0;
        position = 0;
        while (1)
        {
        Tick();
        // Input is reversed to be activated on low, so we need to bitwise inverse PINA to get the
same kind of logic as usual.
        while(!TimerFlag);
        TimerFlag = 0;
        }
        return 0;
}
```