```
/*
 * Buttons.h
 *
 * Created: 5/13/2013 7:24:49 AM
 *  Author: Ariana DeJaco
 */


#ifndef BUTTONS_H_
#define BUTTONS_H_

// The following states are for the buttons
enum Button_States {ButtonOff, ButtonOn} Button1State, Button2State, Button3State;

// The following toggle values are used to drive the speaker
unsigned char Button1ToggleValue, Button2ToggleValue, Button3ToggleValue;

// Define the frequencies for each button
#define Button1Frequency 261.63
#define Button2Frequency 293.66
#define Button3Frequency 329.63

/**********************************************************************/
/* This function represents a Tick on the Button1 state machine.         */
/**********************************************************************/
int Button1Task(int currentState)
{
        unsigned char button1Value = GetBit(PINA, 0);

        Button1State = (enum Button_States) currentState;

        // State Transitions
        switch (Button1State)
        {
                case ButtonOff:
                        if (button1Value == 0)
                                Button1State = ButtonOn;
                        else
                                Button1State = ButtonOff;
                        break;

                case ButtonOn:
                        if (button1Value != 0)
                                Button1State = ButtonOff;
                        else
                                Button1State = ButtonOn;
                        break;

                // Placed for completeness. This should NEVER happen.
                default:
                        Button1State = ButtonOff;
                        break;
        }

        // Action Code: When button is pressed toggle button value which will drive the speaker.
        if (Button1State == ButtonOn)
        {
                if (Button1ToggleValue == 1)
                        Button1ToggleValue = 0;
                else
                        Button1ToggleValue = 1;
        }

        // Return the current state
        return (int) Button1State;
}

/**********************************************************************/
/* This function represents a Tick on the Button2 state machine.         */
/**********************************************************************/
int Button2Task(int currentState)
```

```
{
        unsigned char button2Value = GetBit(PINA, 1);

        Button2State = (enum Button_States) currentState;

        // State Transitions
        switch (Button2State)
        {
                case ButtonOff:
                        if (button2Value != 0)
                                Button2State = ButtonOn;
                        else
                                Button2State = ButtonOff;
                        break;

                case ButtonOn:
                        if (button2Value == 0)
                                Button2State = ButtonOff;
                        else
                                Button2State = ButtonOn;
                        break;

                // Placed for completeness. This should NEVER happen.
                default:
                        Button2State = ButtonOff;
                        break;
        }

        // Action Code: When button is pressed toggle button value which will drive the speaker.
        if (Button2State == ButtonOn)
        {
                if (Button2ToggleValue == 1)
                        Button2ToggleValue = 0;
                else
                        Button2ToggleValue = 1;
        }

        // Return the current state
        return (int) Button2State;
}


/**********************************************************************/
/* This function represents a Tick on the Button3 state machine.         */
/**********************************************************************/
int Button3Task(int currentState)
{
        unsigned char button3Value = GetBit(PINA, 2);

        Button3State = (enum Button_States) currentState;

        // State Transitions
        switch (Button3State)
        {
                case ButtonOff:
                        if (button3Value != 0)
                                Button3State = ButtonOn;
                        else
                                Button3State = ButtonOff;
                        break;

                case ButtonOn:
                        if (button3Value == 0)
                                Button3State = ButtonOff;
                        else
                                Button3State = ButtonOn;
                        break;

                // Placed for completeness. This should NEVER happen.
                default:
                        Button3State = ButtonOff;
                        break;
```

```c
        }

        // Action Code: When button is pressed toggle button value which will drive the speaker.
        if (Button3State == ButtonOn)
        {
                if (Button3ToggleValue == 1)
                        Button3ToggleValue = 0;
                else
                        Button3ToggleValue = 1;
        }

        // Return the current state
        return (int) Button3State;
}

#endif /* BUTTONS_H_ */
```

```
/*
 * KeyPad.h
 *
 * Created: 5/13/2013 6:50:46 AM
 *  Author: Ariana DeJaco
 */


#ifndef KEYPAD_H_
#define KEYPAD_H_

// The following states are for the buttons
enum KeyPad_States {KeyPadOff, Key1, Key2, Key3, Key4, Key5, Key6, Key7, OtherKey} KeyPadState;

#define KEY1_FREQUENCY  261.63
#define KEY2_FREQUENCY  293.66
#define KEY3_FREQUENCY  329.63
#define KEY4_FREQUENCY  349.23
#define KEY5_FREQUENCY  392.00
#define KEY6_FREQUENCY  440.00
#define KEY7_FREQUENCY  493.88

double KeyPadFrequency;

/************************************************************************/
/* Returns ’\0’ if no key pressed, else returns char ’1’, ’2’, ...      */
/* If multiple keys pressed, returns leftmost−topmost one               */
/* Keypad must be connected to port C                                   */
/* Keypad arrangement                                                   */
/*        PC4 PC5 PC6 PC7                                               */
/*   col  1   2   3   4                                                 */
/* row                                                                  */
/* PC0 1   1 | 2 | 3 | A                                                */
/* PC1 2   4 | 5 | 6 | B                                                */
/* PC2 3   7 | 8 | 9 | C                                                */
/* PC3 4   * | 0 | # | D                                                */
/************************************************************************/

unsigned char GetKeypadKey()
{
        PORTC = 0xEF; // Enable col 4 with 0, disable others with 1M−^Rs
        asm("nop"); // add a delay to allow PORTC to stabilize before checking
        if (GetBit(PINC,0)==0) { return(’1’); }
        if (GetBit(PINC,1)==0) { return(’4’); }
        if (GetBit(PINC,2)==0) { return(’7’); }
        if (GetBit(PINC,3)==0) { return(’*’); }

        // Check keys in col 2
        PORTC = 0xDF; // Enable col 5 with 0, disable others with 1M−^Rs
        asm("nop"); // add a delay to allow PORTC to stabilize before checking
        if (GetBit(PINC,0)==0) { return(’2’); }
        if (GetBit(PINC,1)==0) { return(’5’); }
        if (GetBit(PINC,2)==0) { return(’8’); }
        if (GetBit(PINC,3)==0) { return(’0’); }

        // Check keys in col 3
        PORTC = 0xBF; // Enable col 6 with 0, disable others with 1M−^Rs
        asm("nop"); // add a delay to allow PORTC to stabilize before checking
        if (GetBit(PINC,0)==0) { return(’3’); }
        if (GetBit(PINC,1)==0) { return(’6’); }
        if (GetBit(PINC,2)==0) { return(’9’); }
        if (GetBit(PINC,3)==0) { return(’#’); }

        // Check keys in col 4
        PORTC = 0x7F;
        asm("nop"); // add a delay to allow PORTC to stabilize before checking
        if (GetBit(PINC,0)==0) { return(’A’); }
        if (GetBit(PINC,1)==0) { return(’B’); }
        if (GetBit(PINC,2)==0) { return(’C’); }
        if (GetBit(PINC,3)==0) { return(’D’); }

        return(’\0’); // default value
```

```
}

/************************************************************************/
/* This function represents a Tick on the Button1 state machine.        */
/************************************************************************/
int KeyPadTask(int currentState)
{
        // Switching states ONLY depends on the keypad value and not the current state.
        // The state is just the keypad value.
        switch (GetKeypadKey())
        {
                case ’\0’: KeyPadState = KeyPadOff; break;
                case ’1’:  KeyPadState = Key1;      break;
                case ’2’:  KeyPadState = Key2;          break;
                case ’3’:  KeyPadState = Key3;          break;
                case ’4’:  KeyPadState = Key4;          break;
                case ’5’:  KeyPadState = Key5;          break;
                case ’6’:  KeyPadState = Key6;          break;
                case ’7’:  KeyPadState = Key7;          break;
                default:   KeyPadState = OtherKey;      break;
        }

        // Action Code
        switch (KeyPadState)
        {
                case KeyPadOff: KeyPadFrequency = 0.0;          break;
                case Key1:              KeyPadFrequency = KEY1_FREQUENCY;       break;
                case Key2:              KeyPadFrequency = KEY2_FREQUENCY;       break;
                case Key3:              KeyPadFrequency = KEY3_FREQUENCY;       break;
                case Key4:              KeyPadFrequency = KEY4_FREQUENCY;       break;
                case Key5:              KeyPadFrequency = KEY5_FREQUENCY;       break;
                case Key6:              KeyPadFrequency = KEY6_FREQUENCY;       break;
                case Key7:              KeyPadFrequency = KEY7_FREQUENCY;       break;
                case OtherKey:  KeyPadFrequency = 0.0;          break;
        };

        // Return the current state
        return KeyPadState;
}

#endif /* KEYPAD_H_ */
```

```

// Permission to copy is granted provided that this header remains intact.
// This software is provided with no warranties.

////////////////////////////////////////////////////////////////////////////////

#ifndef SCHEDULER_H
#define SCHEDULER_H

////////////////////////////////////////////////////////////////////////////////
//Functionality − finds the greatest common divisor of two values
//Parameter: Two long int’s to find their GCD
//Returns: GCD else 0
unsigned long int findGCD(unsigned long int a, unsigned long int b)
{
        unsigned long int c;
        while(1)
        {
                c = a % b;
                if( c == 0 ) { return b; }
                a = b;
                b = c;
        }
        return 0;
}

////////////////////////////////////////////////////////////////////////////////
//Struct for Tasks represent a running process in our simple real−time operating system
typedef struct _task{
        // Tasks should have members that include: state, period,
        //a measurement of elapsed time, and a function pointer.
        signed   char state;            //Task’s current state
        unsigned long period;           //Task period
        unsigned long elapsedTime;      //Time elapsed since last task tick
        int (*TickFct)(int);            //Task tick function
} task;



#endif //SCHEDULER_H
```

```
/*
 * Speaker_Part1.h
 *
 * Created: 5/13/2013 1:00:48 PM
 *  Author: Ariana DeJaco
 */


#ifndef SPEAKER_PART1_H_
#define SPEAKER_PART1_H_

// The following states are for the speaker
enum    Speaker_States {SpeakerOff, SpeakerOn} SpeakerState;

/************************************************************************/
/* This function will drive the speaker if the Speaker toggle value    */
/* is 0 then D7 is zero otherwise D7 is driven high.                    */
/************************************************************************/
void DriveSpeaker()
{
        // Pulse D7 to drive the speaker
        if (SpeakerState == SpeakerOn)
        {
                if (Button1ToggleValue == 0)
                        PORTD &= 0x7F;                  // Turn OFF D7
                else
                        PORTD |= 0x80;                  // Turn ON D7
        }
        else
        {
                PORTD &= 0x7F;                      // Turn OFF D7
        }
}

/************************************************************************/
/*                                                                      */
/************************************************************************/
void InitSpeaker()
{
        SpeakerState = SpeakerOff;
}

/************************************************************************/
/* This function represents a Tick on the Button1 state machine.        */
/************************************************************************/
int SpeakerTask(int currentState)
{
        SpeakerState = (enum Speaker_States) currentState;

        // State Transitions
        switch (SpeakerState)
        {
                case SpeakerOff:
                        if (Button1State == ButtonOn)
                                SpeakerState = SpeakerOn;
                        break;

                case SpeakerOn:
                        if (Button1State == ButtonOff)
                                SpeakerState = SpeakerOff;
                        break;

                // Placed for completeness. This should NEVER happen.
                default:
                        SpeakerState = SpeakerOff;
                        break;
        }

        // Action Code: The action code is to simply drive the speaker
        DriveSpeaker();

        // Return the current state
        return (int) SpeakerState;
```

```
}

#endif /* SPEAKER_PART1_H_ */
```

```c
/*
 * Speaker_Part2.h
 *
 * Created: 5/13/2013 1:03:49 PM
 *  Author: Ariana DeJaco
 */

#ifndef SPEAKER_PART2_H_
#define SPEAKER_PART2_H_

// The following states are for the speaker
enum    Speaker_States {SpeakerOff, SpeakerOn} SpeakerState;

/************************************************************************/
/* This function will drive the speaker if the Speaker toggle value     */
/* is 0 then D7 is zero otherwise D7 is driven high.                    */
/************************************************************************/
void DriveSpeaker()
{
        unsigned char SpeakerToggleValue;
        if (SpeakerState == SpeakerOn)
        {
                if (Button1State == ButtonOn)
                        SpeakerToggleValue = Button1ToggleValue;
                else if (Button2State == ButtonOn)
                        SpeakerToggleValue = Button2ToggleValue;
                else
                        SpeakerToggleValue = Button3ToggleValue;

                if (SpeakerToggleValue == 0)
                        PORTD &= 0x7F;                  // Turn OFF D7
                else
                        PORTD |= 0x80;                  // Turn ON D7
        }
        else
        {
                PORTD &= 0x7F;                      // Turn OFF D7
        }
}

/************************************************************************/
/*                                                                      */
/************************************************************************/
void InitSpeaker()
{
        SpeakerState = SpeakerOff;
}

/************************************************************************/
/* This function represents a Tick on the Button1 state machine.        */
/************************************************************************/
int SpeakerTask(int currentState)
{
        SpeakerState = (enum Speaker_States) currentState;

        // State Transitions
        switch (SpeakerState)
        {
                case SpeakerOff:
                        if (((Button1State == ButtonOn)  && (Button2State == ButtonOff) && (Button3State == ButtonOff)) ||
                                ((Button1State == ButtonOff) && (Button2State == ButtonOn)  && (Button3State == ButtonOff)) ||
                                ((Button1State == ButtonOff) && (Button2State == ButtonOff) && (Button3State == ButtonOn)))
                                SpeakerState = SpeakerOn;
                        break;

                case SpeakerOn:
                        if (! (((Button1State == ButtonOn)  && (Button2State == ButtonOff) && (Button3State == ButtonOff)) ||
                                   ((Button1State == ButtonOff) && (Button2State == ButtonOn)  && (Button3State == ButtonOff)) ||
```

```c
                                   ((Button1State == ButtonOff) && (Button2State == ButtonOff) && (Button3State == ButtonOn))))
                                SpeakerState = SpeakerOff;
                        break;

                // Placed for completeness. This should NEVER happen.
                default:
                        SpeakerState = SpeakerOff;
                        break;
        }

        // Action Code: The action code is to simply drive the speaker
        DriveSpeaker();

        // Return the current state
        return (int) SpeakerState;
}

#endif /* SPEAKER_PART2_H_ */
```

```
/*
 * Speaker_Part3.h
 *
 * Created: 5/13/2013 1:08:28 PM
 *  Author: Ariana DeJaco
 */

#ifndef SPEAKER_PART3_H_
#define SPEAKER_PART3_H_

// The following states are for the speaker
enum    Speaker_States {SpeakerOff, SpeakerOn} SpeakerState;

/************************************************************************/
/* This code is for PART 3 of the lab and is used to configures the D7  */
/* to toggle automatically using Timer2 of ATMega32.                    */
/* Timer2 Page 125: WGM21 = 1, WGM20 = 0 is CTC mode.                   */
/*             CTC mode is clear timer on compare match. (page 119)     */
/* Timer2 Page 126: COM21 = 0, COM20 = 1 is Toggle on compare.          */
/* Timer2 Page 127: CS22 = 1, CS21 = 0, CS20 = 0 −> Clk / 64.           */
/************************************************************************/
void InitPWM()
{
        TCCR2 = (1 << WGM21) | (1 << COM20) | (1 << CS22) ;
}

/************************************************************************/
/* Sets the frequency of the Toggle Pin.                                */
/* The correct equation is on of page 120. If you solve for OCRn you    */
/* get the equation listed below for converting freq to OCR2.           */
/************************************************************************/
void set_PWM (double frequency)
{
        if (frequency < 1)
        OCR2 = 0;
        else
        OCR2 = (unsigned char) ((8000000.0 / (128.0 * frequency)) − 1.0);
}

/************************************************************************/
/* This function will drive the speaker if the Speaker toggle value     */
/* is 0 then D7 is zero otherwise D7 is driven high.                    */
/************************************************************************/
void DriveSpeaker()
{
        if (SpeakerState == SpeakerOn)
        {
                // Select the frequency based on the button pressed
                if (Button1State == ButtonOn)
                {
                        set_PWM(Button1Frequency);
                }
                else
                {
                        set_PWM(0.0);
                }
        }
        else
        {
                set_PWM(0.0);
        }
}

/************************************************************************/
/*                                                                      */
/************************************************************************/
void InitSpeaker()
{
        SpeakerState = SpeakerOff;
        InitPWM();
}


```

```
/************************************************************************/
/* This function represents a Tick on the Button1 state machine.        */
/************************************************************************/
int SpeakerTask(int currentState)
{
        SpeakerState = (enum Speaker_States) currentState;

        // State Transitions
        switch (SpeakerState)
        {
                case SpeakerOff:
                        if (Button1State == ButtonOn)
                                SpeakerState = SpeakerOn;
                        break;

                case SpeakerOn:
                        if (Button1State == ButtonOff)
                                SpeakerState = SpeakerOff;
                        break;

                // Placed for completeness. This should NEVER happen.
                default:
                        SpeakerState = SpeakerOff;
                        break;
        }

        // Action Code: The action code is to simply drive the speaker
        DriveSpeaker();

        // Return the current state
        return (int) SpeakerState;
}

#endif /* SPEAKER_PART3_H_ */
```

```
/*
 * Speaker_Part4.h
 *
 * Created: 5/13/2013 1:19:05 PM
 *  Author: Ariana DeJaco
 */


#ifndef SPEAKER_PART4_H_
#define SPEAKER_PART4_H_

#define DEBOUNCE_COUNT_LIMIT 200

// The following states are for the speaker
enum    Speaker_States {SpeakerOff, SpeakerOn, SpeakerOnWaitForRelease, SpeakerOffWaitForRelease} SpeakerState;
double  SpeakerFrequency;
unsigned char   DebounceCounter;
int     ToneTimer;

/************************************************************************/
/* This code is for PART 3 of the lab and is used to configures the D7  */
/* to toggle automatically using Timer2 of ATMega32.                    */
/* Timer2 Page 125: WGM21 = 1, WGM20 = 0 is CTC mode.                   */
/*              CTC mode is clear timer on compare match. (page 119)      */
/* Timer2 Page 126: COM21 = 0, COM20 = 1 is Toggle on compare.          */
/* Timer2 Page 127: CS22 = 1, CS21 = 0, CS20 = 0 -> Clk / 64.           */
/************************************************************************/
void InitPWM()
{
        TCCR2 = (1 << WGM21) | (1 << COM20) | (1 << CS22) ;
}

/************************************************************************/
/* Sets the frequency of the Toggle Pin.                                */
/* The correct equation is on of page 120. If you solve for OCRn you    */
/* get the equation listed below for converting freq to OCR2.           */
/************************************************************************/
void set_PWM (double frequency)
{
        if (frequency < 1)
        OCR2 = 0;
        else
        OCR2 = (unsigned char) ((8000000.0 / (128.0 * frequency)) - 1.0);
}

/************************************************************************/
/* This function will drive the speaker if the Speaker toggle value     */
/* is 0 then D7 is zero otherwise D7 is driven high.                    */
/************************************************************************/
void DriveSpeaker()
{
        if ((SpeakerState == SpeakerOn) || (SpeakerState == SpeakerOnWaitForRelease))
        {
                // Select the frequency based on the button pressed
                if ((Button2State == ButtonOn) && (Button3State == ButtonOff))
                {
                        SpeakerFrequency += 0.001;
                }
                else if ((Button2State == ButtonOff) && (Button3State == ButtonOn))
                {
                        SpeakerFrequency -= 0.001;
                }
                else
                        SpeakerFrequency = SpeakerFrequency; // Don’t change the frequency....

                // Drive the speaker with this frequency
                set_PWM(SpeakerFrequency);
        }
        else
```

```
        {
                set_PWM(0.0);
        }
}

/************************************************************************/
/*                                                                      */
/************************************************************************/
void InitSpeaker()
{
        SpeakerState = SpeakerOff;
        DebounceCounter  = 0;
        SpeakerFrequency = 349.23;
        InitPWM();
}


/************************************************************************/
/* This function represents a Tick on the Button1 state machine.        */
/************************************************************************/
int SpeakerTask(int currentState)
{
        SpeakerState = (enum Speaker_States) currentState;

        // State Transitions
        switch (SpeakerState)
        {
                case SpeakerOff:
                        if (Button1State == ButtonOn)
                        {
                                SpeakerState = SpeakerOnWaitForRelease;
                                DebounceCounter = 0;
                        }
                        break;

                case SpeakerOn:
                        if (Button1State == ButtonOn)
                        {
                                SpeakerState = SpeakerOffWaitForRelease;
                                DebounceCounter = 0;
                        }
                        break;

                case SpeakerOffWaitForRelease:
                        if (DebounceCounter < DEBOUNCE_COUNT_LIMIT)
                                DebounceCounter++;  // Wait until we hit DEBOUNCE_COUNT_LIMIT
                        else
                        {
                                if (Button1State == ButtonOff)
                                {
                                        SpeakerState = SpeakerOff;
                                }
                                else
                                        DebounceCounter = 0;    // Wait some more...
                        }
                        break;

                case SpeakerOnWaitForRelease:
                        if (DebounceCounter < DEBOUNCE_COUNT_LIMIT)
                                DebounceCounter++;  // Wait until we hit DEBOUNCE_COUNT_LIMIT
                        else
                        {
                                if (Button1State == ButtonOff)
                                {
                                        SpeakerState = SpeakerOn;
                                }
                                else
                                        DebounceCounter = 0;    // Wait some more....
                        }
                        break;

                // Placed for completeness. This should NEVER happen.
                default:
                        SpeakerState = SpeakerOff;
                        break;
        }

        // Action Code: The action code is to simply drive the speaker
        DriveSpeaker();

        // Return the current state
        return (int) SpeakerState;
}

#endif /* SPEAKER_PART4_H_ */
```

```
/*
 * Speaker_PartChallange.h
 *
 * Created: 5/13/2013 1:55:57 PM
 *  Author: Ariana DeJaco
 */

#ifndef SPEAKER_PARTCHALLANGE_H_
#define SPEAKER_PARTCHALLANGE_H_

// The following states are for the speaker
enum Speaker_States {SpeakerOff, SpeakerOn} SpeakerState;

/************************************************************************/
/* This code is for PART 3 of the lab and is used to configures the D7  */
/* to toggle automatically using Timer2 of ATMega32.                    */
/* Timer2 Page 125: WGM21 = 1, WGM20 = 0 is CTC mode.                   */
/*             CTC mode is clear timer on compare match. (page 119)     */
/* Timer2 Page 126: COM21 = 0, COM20 = 1 is Toggle on compare.          */
/* Timer2 Page 127: CS22 = 1, CS21 = 0, CS20 = 0 -> Clk / 64.           */
/************************************************************************/
void InitPWM()
{
        TCCR2 = (1 << WGM21) | (1 << COM20) | (1 << CS22) ;
}

/************************************************************************/
/* Sets the frequency of the Toggle Pin.                                */
/* The correct equation is on of page 120. If you solve for OCRn you    */
/* get the equation listed below for converting freq to OCR2.           */
/************************************************************************/
void set_PWM (double frequency)
{
        if (frequency < 1)
                OCR2 = 0;
        else
                OCR2 = (unsigned char) ((8000000.0 / (128.0 * frequency)) - 1.0);
}

/************************************************************************/
/* This function will drive the speaker if the Speaker toggle value     */
/* is 0 then D7 is zero otherwise D7 is driven high.                    */
/************************************************************************/
void DriveSpeaker()
{
        if (SpeakerState == SpeakerOn)
        {
                set_PWM(KeyPadFrequency);
        }
        else
        {
                set_PWM(0.0);
        }
}

/************************************************************************/
/*                                                                      */
/************************************************************************/
void InitSpeaker()
{
        SpeakerState = SpeakerOff;
        InitPWM();
}

/************************************************************************/
/* This function represents a Tick on the Button1 state machine.        */
/************************************************************************/
int SpeakerTask(int currentState)
{
        SpeakerState = (enum Speaker_States) currentState;

        // State Transitions
```

```
        switch (SpeakerState)
        {
                case SpeakerOff:
                        if ((KeyPadState != KeyPadOff) && (KeyPadState != OtherKey))
                                SpeakerState = SpeakerOn;
                        break;

                case SpeakerOn:
                        if ((KeyPadState == KeyPadOff) || (KeyPadState == OtherKey))
                                SpeakerState = SpeakerOff;
                        break;

                // Placed for completeness. This should NEVER happen.
                default:
                        SpeakerState = SpeakerOff;
                        break;
        }

        // Action Code: The action code is to simply drive the speaker
        DriveSpeaker();

        // Return the current state
        return (int) SpeakerState;
}

#endif /* SPEAKER_PARTCHALLANGE_H_ */
```

```

// Permission to copy is granted provided that this header remains intact.
// This software is provided with no warranties.

////////////////////////////////////////////////////////////////////////////////

#ifndef TIMER_H
#define TIMER_H

#include <avr/interrupt.h>

volatile unsigned char TimerFlag = 0; // TimerISR() sets this to 1. C programmer should clear to 0.

// Internal variables for mapping AVR’s ISR to our cleaner TimerISR model.
unsigned long _avr_timer_M = 1; // Start count from here, down to 0. Default 1ms
unsigned long _avr_timer_cntcurr = 0; // Current internal count of 1ms ticks

// Set TimerISR() to tick every M ms
void TimerSet(unsigned long M) {
        _avr_timer_M = M;
        _avr_timer_cntcurr = _avr_timer_M;
}

void TimerOn() {
        // AVR timer/counter controller register TCCR0
        TCCR0   = 0x0A; // bit3bit6=10: CTC mode (clear timer on compare)
        // bit2bit1bit0=010: prescaler /8
        // 00001010: 0x0A
        // SO, 8 MHz clock or 8,000,000 /8 = 1,000,000 ticks/s
        // Thus, TCNT0 register will count at 1,000,000 ticks/s

        // AVR output compare register OCR0.
        OCR0    = 100;  // Timer interrupt will be generated when TCNT0==OCR0
        TIMSK   = 0x02; // bit1: OCIE0 −− enables compare match interrupt
                                        // 1,000,000 / 100 = 0.00001

        //Initialize avr counter
        TCNT0 = 0;

        // TimerISR will be called every _avr_timer_cntcurr milliseconds
        _avr_timer_cntcurr = _avr_timer_M;

        //Enable global interrupts
        SREG |= 0x80;   // 0x80: 1000000
}

void TimerOff() {
        TCCR0   = 0x00; // bit2bit1bit0=000: timer off
}

void TimerISR() {
        TimerFlag = 1;
}

// In our approach, the C programmer does not touch this ISR, but rather TimerISR()
ISR(TIMER0_COMP_vect)
{
        // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn settings)
        _avr_timer_cntcurr−−;                   // Count down to 0 rather than up to TOP
        if (_avr_timer_cntcurr == 0) {  // results in a more efficient compare
                TimerISR();                             // Call the ISR that the user uses
                _avr_timer_cntcurr = _avr_timer_M;
        }
}

#endif //TIMER_H
```

```c
/*     adeja001_lab9_part1.c − 5−13−13
 *    Name & E−mail: Ariana Dejaco adeja001@ucr.edu
 *    CS Login: adeja001
 *    Partner(s) Name & E−mail: Joshua DeForest−Williams jdefo002@ucr.edu
 *    Lab Section: 22
 *    Assignment: Lab # 9 Exercise # 1
 *    Exercise Description: The goal of this part of the lab is to produce
 *    a middle C on the speaker when a button is pressed and held.
 *    I acknowledge all content contained herein, excluding template or example
 *    code, is my own original work.
 */
#include <avr/io.h>
#include <avr/sfr_defs.h>
#include <math.h>
#include "Bit.h"
#include "Buttons.h"
#include "KeyPad.h"
#include "Speaker_Part1.h"
#include "Timer0.h"
#include "Schedular.h"

/**********************************************************************/
/* Initialize the ports                                               */
/**********************************************************************/
void InitPorts()
{
        // Make PORTD7 as an output. This is connected to the speaker
        DDRD  = 0x80;
        PORTD = 0x7F;  // Do NOT use pull up resistor on
        // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
        // and PC[7:4] are outputs.
        DDRC  = 0xF0;
        PORTC = 0x0F;
        // Make Port A connected to the buttons. Make them all inputs
        DDRA = 0x00;
}

/**********************************************************************/
/* This function calculates the frequency used. The math is:          */
/* T = 1.0 / frequency − Converts to Period                           */
/* Ticks = T * 10,000 − Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
/* Toggle Rate = Ticks / 2.                                           */
/**********************************************************************/
unsigned long calculateFrequencyTick(double frequency)
{
        double Period_F = 1.0 / frequency;                                   // Period = 1 / Frequency
        double TicksPerSecond_F = Period_F * 10000.0;                  // Period * 10,000 ticks/second = Ticks.
        long   TicksPerSecond_L = (long) TicksPerSecond_F;             // Convert to a long from a float
        long   ToggleRate = TicksPerSecond_L / 2;                             // It takes two toggle to make 1 period
        return ToggleRate;
}

/**********************************************************************/
/* Initialize Task Scheduler                                          */
/**********************************************************************/
void RunTaskScheduler()
{
        unsigned long int GCD = 1;

        // Declare the local variables. The following are the ticks that the
        // button and keypads need to run in. Initially they are all zero (disabled)
        // They are added depending on which part of the lab we are running.
        unsigned long int Button1Tick = 0;  // Initially disabled
        unsigned long int Button2Tick = 0;  // Initially disabled
        unsigned long int Button3Tick = 0;  // Initially disabled
        unsigned long int KeypadTick  = 0;  // Initially disabled

```

```c
        ///////////////////////////////////////////////////////////////////////////////
        // Determine the real ticks that we need to run the button and keypad − If any..
        // Based on the lab part that we are running.
        ///////////////////////////////////////////////////////////////////////////////
#ifdef SPEAKER_PART1_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
#endif
#ifdef SPEAKER_PART2_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
        Button2Tick = calculateFrequencyTick(Button2Frequency);
        Button3Tick = calculateFrequencyTick(Button3Frequency);
#endif
#ifdef SPEAKER_PART3_H_
        Button1Tick = 100;   // Good number for button sampling....
#endif
#ifdef SPEAKER_PART4_H_
        Button1Tick = 100;   // Good number for button sampling....
        Button2Tick = 10;    // Sounds good for increasing freq
        Button3Tick = 10;    // Sounds good for decreasing freq
#endif
#ifdef SPEAKER_PARTCHALLANGE_H_
        KeypadTick  = 100;   // Good number for button sampling....
#endif

        //Recalculate GCD periods for scheduler
        unsigned long int SpeakerPeriod = GCD;          // Speaker must be driven at the fastest rate (Twice the fastest button).
        unsigned long int Button1Period = Button1Tick/GCD;
        unsigned long int Button2Period = Button2Tick/GCD;
        unsigned long int Button3Period = Button3Tick/GCD;
        unsigned long int KeyPadPeriod  = KeypadTick/GCD;

        // Declare an array of tasks
        static task task1, task2, task3, task4, task5;
        task *tasks[] = { &task1, &task2, &task3, &task4, &task5};
        const unsigned short numTasks = sizeof(tasks)/sizeof(task*);

        // Task 1
        task1.state = (int) SpeakerOff;     //Task initial state.
        task1.period = SpeakerPeriod;       //Task Period.
        task1.elapsedTime = SpeakerPeriod;  //Task current elapsed time.
        task1.TickFct = &SpeakerTask;       //Function pointer for the tick.

        // Task 2
        task2.state = (int) ButtonOff;     //Task initial state.
        task2.period = Button1Period;       //Task Period.
        task2.elapsedTime = Button1Period;  //Task current elapsed time.
        task2.TickFct = &Button1Task;       //Function pointer for the tick.

        // Task 3
        task3.state = (int) ButtonOff;     //Task initial state.
        task3.period = Button2Period;       //Task Period.
        task3.elapsedTime = Button2Period;  //Task current elapsed time.
        task3.TickFct = &Button2Task;       //Function pointer for the tick.

        // Task 4
        task4.state = (int) ButtonOff;     //Task initial state.
        task4.period = Button3Period;       //Task Period.
        task4.elapsedTime = Button3Period;  //Task current elapsed time.
        task4.TickFct = &Button3Task;       //Function pointer for the tick.

        // Task 5
        task5.state = (int) KeyPadOff;      //Task initial state.
        task5.period = KeyPadPeriod;        //Task Period.
        task5.elapsedTime = KeyPadPeriod;   //Task current elapsed time.
        task5.TickFct = &KeyPadTask;        //Function pointer for the tick.

        // Set the timer and turn it on
        TimerSet(GCD);
        TimerOn();

        // Run all tasks forever....
        unsigned short i; // Scheduler for-loop iterator
        while(1)
        {
                // Scheduler code
                for ( i = 0; i < numTasks; i++ )
                {
                        if (tasks[i]->period != 0)
                        {
                                // Task is ready to tick
                                if ( tasks[i]->elapsedTime == tasks[i]->period )
                                {
                                        // Setting next state for task
                                        tasks[i]->state = tasks[i]->TickFct(tasks[i]->state);
                                        // Reset the elapsed time for next tick.
                                        tasks[i]->elapsedTime = 0;
                                }
                                tasks[i]->elapsedTime += 1;
                        }
                }

                // Process_LCD_Task(1);
                while(!TimerFlag);
                TimerFlag = 0;
        }
}

int main(void)
{
   InitPorts();
   InitSpeaker();
   RunTaskScheduler();
}
```

```c
/*    adeja001_lab9_part2.c − 5−13−13
 *    Name & E−mail: Ariana Dejaco adeja001@ucr.edu
 *    CS Login: adeja001
 *    Partner(s) Name & E−mail: Joshua DeForest−Williams jdefo002@ucr.edu
 *    Lab Section: 22
 *    Assignment: Lab # 9 Exercise # 2
 *    Exercise Description: For this part of the lab, create a system that
 *    has three button inputs and, depending on which button is pressed,
 *    the speaker will output either a middle C, D, or E.
 *
 *    I acknowledge all content contained herein, excluding template or example
 *    code, is my own original work.
 */
#include <avr/io.h>
#include <avr/sfr_defs.h>
#include <math.h>
#include "Bit.h"
#include "Buttons.h"
#include "KeyPad.h"
#include "Speaker_Part2.h"
#include "Timer0.h"
#include "Schedular.h"

/*************************************************************************/
/* Initialize the ports                                                  */
/*************************************************************************/
void InitPorts()
{
        // Make PORTD7 as an output. This is connected to the speaker
        DDRD  = 0x80;
        PORTD = 0x7F;  // Do NOT use pull up resistor on
        // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
        // and PC[7:4] are outputs.
        DDRC  = 0xF0;
        PORTC = 0x0F;
        // Make Port A connected to the buttons. Make them all inputs
        DDRA = 0x00;
}

/*************************************************************************/
/* This function calculates the frequency used. The math is:             */
/* T = 1.0 / frequency − Converts to Period                              */
/* Ticks = T * 10,000 − Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
/* Toggle Rate = Ticks / 2.                                              */
/*************************************************************************/
unsigned long calculateFrequencyTick(double frequency)
{
        double Period_F = 1.0 / frequency;                              // Period = 1 / Frequency
        double TicksPerSecond_F = Period_F * 10000.0;                   // Period * 10,000 ticks/second = Ticks.
        long   TicksPerSecond_L = (long) TicksPerSecond_F;              // Convert to a long from a float
        long   ToggleRate = TicksPerSecond_L / 2;                       // It takes two toggle to make 1 period
        return ToggleRate;
}

/*************************************************************************/
/* Initialize Task Scheduler                                             */
/*************************************************************************/
void RunTaskScheduler()
{
        unsigned long int GCD = 1;

        // Declare the local variables. The following are the ticks that the
        // button and keypads need to run in. Initially they are all zero (disabled)
        // They are added depending on which part of the lab we are running.
        unsigned long int Button1Tick = 0;  // Initially disabled
        unsigned long int Button2Tick = 0;  // Initially disabled
        unsigned long int Button3Tick = 0;  // Initially disabled
```

```c
        unsigned long int KeypadTick  = 0;  // Initially disabled

        ///////////////////////////////////////////////////////////////////////////////
        // Determine the real ticks that we need to run the button and keypad − If any..
        // Based on the lab part that we are running.
        ///////////////////////////////////////////////////////////////////////////////
#ifdef SPEAKER_PART1_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
#endif
#ifdef SPEAKER_PART2_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
        Button2Tick = calculateFrequencyTick(Button2Frequency);
        Button3Tick = calculateFrequencyTick(Button3Frequency);
#endif
#ifdef SPEAKER_PART3_H_
        Button1Tick = 100;   // Good number for button sampling....
#endif
#ifdef SPEAKER_PART4_H_
        Button1Tick = 100;   // Good number for button sampling....
        Button2Tick = 10;    // Sounds good for increasing freq
        Button3Tick = 10;    // Sounds good for decreasing freq
#endif
#ifdef SPEAKER_PARTCHALLANGE_H_
        KeypadTick  = 100;   // Good number for button sampling....
#endif

        //Recalculate GCD periods for scheduler
        unsigned long int SpeakerPeriod = GCD;          // Speaker must be driven at the fastest rate (Twice the fastest button).
        unsigned long int Button1Period = Button1Tick/GCD;
        unsigned long int Button2Period = Button2Tick/GCD;
        unsigned long int Button3Period = Button3Tick/GCD;
        unsigned long int KeyPadPeriod  = KeypadTick/GCD;

        // Declare an array of tasks
        static task task1, task2, task3, task4, task5;
        task *tasks[] = { &task1, &task2, &task3, &task4, &task5};
        const unsigned short numTasks = sizeof(tasks)/sizeof(task*);

        // Task 1
        task1.state = (int) SpeakerOff;     //Task initial state.
        task1.period = SpeakerPeriod;       //Task Period.
        task1.elapsedTime = SpeakerPeriod;  //Task current elapsed time.
        task1.TickFct = &SpeakerTask;       //Function pointer for the tick.

        // Task 2
        task2.state = (int) ButtonOff;     //Task initial state.
        task2.period = Button1Period;       //Task Period.
        task2.elapsedTime = Button1Period;  //Task current elapsed time.
        task2.TickFct = &Button1Task;       //Function pointer for the tick.

        // Task 3
        task3.state = (int) ButtonOff;     //Task initial state.
        task3.period = Button2Period;       //Task Period.
        task3.elapsedTime = Button2Period;  //Task current elapsed time.
        task3.TickFct = &Button2Task;       //Function pointer for the tick.

        // Task 4
        task4.state = (int) ButtonOff;     //Task initial state.
        task4.period = Button3Period;       //Task Period.
        task4.elapsedTime = Button3Period;  //Task current elapsed time.
        task4.TickFct = &Button3Task;       //Function pointer for the tick.

        // Task 5
        task5.state = (int) KeyPadOff;      //Task initial state.
        task5.period = KeyPadPeriod;        //Task Period.
        task5.elapsedTime = KeyPadPeriod;   //Task current elapsed time.
        task5.TickFct = &KeyPadTask;        //Function pointer for the tick.

        // Set the timer and turn it on
        TimerSet(GCD);
        TimerOn();

        // Run all tasks forever....
        unsigned short i; // Scheduler for-loop iterator
        while(1)
        {
                // Scheduler code
                for ( i = 0; i < numTasks; i++ )
                {
                        if (tasks[i]->period != 0)
                        {
                                // Task is ready to tick
                                if ( tasks[i]->elapsedTime == tasks[i]->period )
                                {
                                        // Setting next state for task
                                        tasks[i]->state = tasks[i]->TickFct(task
s[i]->state);
                                        // Reset the elapsed time for next tick.
                                        tasks[i]->elapsedTime = 0;
                                }
                                tasks[i]->elapsedTime += 1;
                        }
                }

                // Process_LCD_Task(1);
                while(!TimerFlag);
                TimerFlag = 0;
        }
}

int main(void)
{
   InitPorts();
   InitSpeaker();
   RunTaskScheduler();
}
```

```
/*     adeja001_lab9_part3.c − 5−13−13
 *    Name & E−mail: Ariana Dejaco adeja001@ucr.edu
 *    CS Login: adeja001
 *    Partner(s) Name & E−mail: Joshua DeForest−Williams jdefo002@ucr.edu
 *    Lab Section: 22
 *    Assignment: Lab # 9 Exercise # 3
 *    Exercise Description:This section will be using the PWM functionality
 *    on timer/counter 2 of the ATmega32. Since we will be using a different
 *    timer than the one used in previous labs, the changes made to TCCR0
 *    and OCR0 can be reverted back to their original values.
 *
 *    I acknowledge all content contained herein, excluding template or example
 *    code, is my own original work.
 */

#include <avr/io.h>
#include <avr/sfr_defs.h>
#include <math.h>
#include "Bit.h"
#include "Buttons.h"
#include "KeyPad.h"
#include "Speaker_Part3.h"
#include "Timer0.h"
#include "Schedular.h"

/*************************************************************************/
/* Initialize the ports                                                  */
/*************************************************************************/
void InitPorts()
{
        // Make PORTD7 as an output. This is connected to the speaker
        DDRD  = 0x80;
        PORTD = 0x7F;  // Do NOT use pull up resistor on
        // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
        // and PC[7:4] are outputs.
        DDRC  = 0xF0;
        PORTC = 0x0F;
        // Make Port A connected to the buttons. Make them all inputs
        DDRA = 0x00;
}

/*************************************************************************/
/* This function calculates the frequency used. The math is:             */
/* T = 1.0 / frequency − Converts to Period                              */
/* Ticks = T * 10,000 − Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
/* Toggle Rate = Ticks / 2.                                              */
/*************************************************************************/
unsigned long calculateFrequencyTick(double frequency)
{
        double Period_F = 1.0 / frequency;                                                          // Period = 1 / Frequency
        double TicksPerSecond_F = Period_F * 10000.0;                   // Period * 10,000 ticks/second = Ticks.
        long   TicksPerSecond_L = (long) TicksPerSecond_F;              // Convert to a long from a float
        long   ToggleRate = TicksPerSecond_L / 2;                                               // It takes two toggle to make 1 period
        return ToggleRate;
}

/*************************************************************************/
/* Initialize Task Scheduler                                             */
/*************************************************************************/
void RunTaskScheduler()
{
        unsigned long int GCD = 1;

        // Declare the local variables. The following are the ticks that the
        // button and keypads need to run in. Initially they are all zero (disabled)
        // They are added depending on which part of the lab we are running.
        unsigned long int Button1Tick = 0;  // Initially disabled
```

```
        unsigned long int Button2Tick = 0;  // Initially disabled
        unsigned long int Button3Tick = 0;  // Initially disabled
        unsigned long int KeypadTick  = 0;  // Initially disabled

        ///////////////////////////////////////////////////////////////////////////////
        // Determine the real ticks that we need to run the button and keypad − If any..
        // Based on the lab part that we are running.
        ///////////////////////////////////////////////////////////////////////////////
#ifdef SPEAKER_PART1_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
#endif
#ifdef SPEAKER_PART2_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
        Button2Tick = calculateFrequencyTick(Button2Frequency);
        Button3Tick = calculateFrequencyTick(Button3Frequency);
#endif
#ifdef SPEAKER_PART3_H_
        Button1Tick = 100;   // Good number for button sampling....
#endif
#ifdef SPEAKER_PART4_H_
        Button1Tick = 100;   // Good number for button sampling....
        Button2Tick = 10;    // Sounds good for increasing freq
        Button3Tick = 10;    // Sounds good for decreasing freq
#endif
#ifdef SPEAKER_PARTCHALLANGE_H_
        KeypadTick  = 100;   // Good number for button sampling....
#endif

        //Recalculate GCD periods for scheduler
        unsigned long int SpeakerPeriod = GCD;          // Speaker must be driven at the fastest rate (Twice the fastest button).
        unsigned long int Button1Period = Button1Tick/GCD;
        unsigned long int Button2Period = Button2Tick/GCD;
        unsigned long int Button3Period = Button3Tick/GCD;
        unsigned long int KeyPadPeriod  = KeypadTick/GCD;

        // Declare an array of tasks
        static task task1, task2, task3, task4, task5;
        task *tasks[] = { &task1, &task2, &task3, &task4, &task5};
        const unsigned short numTasks = sizeof(tasks)/sizeof(task*);

        // Task 1
        task1.state = (int) SpeakerOff;     //Task initial state.
        task1.period = SpeakerPeriod;       //Task Period.
        task1.elapsedTime = SpeakerPeriod;  //Task current elapsed time.
        task1.TickFct = &SpeakerTask;       //Function pointer for the tick.

        // Task 2
        task2.state = (int) ButtonOff;     //Task initial state.
        task2.period = Button1Period;       //Task Period.
        task2.elapsedTime = Button1Period;  //Task current elapsed time.
        task2.TickFct = &Button1Task;       //Function pointer for the tick.

        // Task 3
        task3.state = (int) ButtonOff;     //Task initial state.
        task3.period = Button2Period;       //Task Period.
        task3.elapsedTime = Button2Period;  //Task current elapsed time.
        task3.TickFct = &Button2Task;       //Function pointer for the tick.

        // Task 4
        task4.state = (int) ButtonOff;     //Task initial state.
        task4.period = Button3Period;       //Task Period.
        task4.elapsedTime = Button3Period;  //Task current elapsed time.
        task4.TickFct = &Button3Task;       //Function pointer for the tick.

        // Task 5
        task5.state = (int) KeyPadOff;      //Task initial state.
        task5.period = KeyPadPeriod;        //Task Period.
        task5.elapsedTime = KeyPadPeriod;   //Task current elapsed time.
        task5.TickFct = &KeyPadTask;        //Function pointer for the tick.
```

```c

        // Set the timer and turn it on
        TimerSet(GCD);
        TimerOn();

        // Run all tasks forever....
        unsigned short i; // Scheduler for-loop iterator
        while(1)
        {
                // Scheduler code
                for ( i = 0; i < numTasks; i++ )
                {
                        if (tasks[i]->period != 0)
                        {
                                // Task is ready to tick
                                if ( tasks[i]->elapsedTime == tasks[i]->period )
                                {
                                        // Setting next state for task
                                        tasks[i]->state = tasks[i]->TickFct(tasks[i]->state);
                                        // Reset the elapsed time for next tick.
                                        tasks[i]->elapsedTime = 0;
                                }
                                tasks[i]->elapsedTime += 1;
                        }
                }

                // Process_LCD_Task(1);
                while(!TimerFlag);
                TimerFlag = 0;
        }
}

int main(void)
{
   InitPorts();
   InitSpeaker();
   RunTaskScheduler();
}
```

```c
/*    adeja001_lab9_part4.c - 5-13-13
 *    Name & E-mail: Ariana Dejaco Adeja001@ucr.edu
 *    CS Login: adeja001
 *    Partner(s) Name & E-mail: Joshua DeForest-Williams jdefo002@ucr.edu
 *    Lab Section: 22
 *    Assignment: Lab # 9  Exercise # 4 Challenge
 *    Exercise Description: Expand upon part 3 of the lab by adding three
 *    additional buttons with the following functionality. One button will
 *    toggle sound on/off. The other two buttons will be used to raise or
 *    lower the tone. Holding the tone buttons down should raise/lower the
 *    tone slowly (A note every second will be slow enough).
 *
 *    I acknowledge all content contained herein, excluding template or example
 *    code, is my own original work.
 */

#include <avr/io.h>
#include <avr/sfr_defs.h>
#include <math.h>
#include "Bit.h"
#include "Buttons.h"
#include "KeyPad.h"
#include "Speaker_Part4.h"
#include "Timer0.h"
#include "Schedular.h"

/************************************************************************/
/* Initialize the ports                                                 */
/************************************************************************/
void InitPorts()
{
        // Make PORTD7 as an output. This is connected to the speaker
        DDRD  = 0x80;
        PORTD = 0x7F;  // Do NOT use pull up resistor on
        // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
        // and PC[7:4] are outputs.
        DDRC  = 0xF0;
        PORTC = 0x0F;
        // Make Port A connected to the buttons. Make them all inputs
        DDRA = 0x00;
}

/************************************************************************/
/* This function calculates the frequency used. The math is:                    */
/* T = 1.0 / frequency - Converts to Period                             */
/* Ticks = T * 10,000 - Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
/* Toggle Rate = Ticks / 2.                                             */
/************************************************************************/
unsigned long calculateFrequencyTick(double frequency)
{
        double Period_F = 1.0 / frequency;                                      // Period = 1 / Frequency
        double TicksPerSecond_F = Period_F * 10000.0;                   // Period * 10,000 ticks/second = Ticks.
        long   TicksPerSecond_L = (long) TicksPerSecond_F;              // Convert to a long from a float
        long   ToggleRate = TicksPerSecond_L / 2;                               // It takes two toggle to make 1 period
        return ToggleRate;
}

/************************************************************************/
/* Initialize Task Scheduler                                            */
/************************************************************************/
void RunTaskScheduler()
{
        unsigned long int GCD = 1;

        // Declare the local variables. The following are the ticks that the
        // button and keypads need to run in. Initially they are all zero (disabled)
        // They are added depending on which part of the lab we are running.
```

```c
        unsigned long int Button1Tick = 0;  // Initially disabled
        unsigned long int Button2Tick = 0;  // Initially disabled
        unsigned long int Button3Tick = 0;  // Initially disabled
        unsigned long int KeypadTick  = 0;  // Initially disabled

        ////////////////////////////////////////////////////////////////////////////////
        // Determine the real ticks that we need to run the button and keypad - If any..
        // Based on the lab part that we are running.
        ////////////////////////////////////////////////////////////////////////////////
#ifdef SPEAKER_PART1_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
#endif
#ifdef SPEAKER_PART2_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
        Button2Tick = calculateFrequencyTick(Button2Frequency);
        Button3Tick = calculateFrequencyTick(Button3Frequency);
#endif
#ifdef SPEAKER_PART3_H_
        Button1Tick = 100;   // Good number for button sampling....
#endif
#ifdef SPEAKER_PART4_H_
        Button1Tick = 100;   // Good number for button sampling....
        Button2Tick = 10;    // Sounds good for increasing freq
        Button3Tick = 10;    // Sounds good for decreasing freq
#endif
#ifdef SPEAKER_PARTCHALLANGE_H_
        KeypadTick  = 100;   // Good number for button sampling....
#endif

        //Recalculate GCD periods for scheduler
        unsigned long int SpeakerPeriod = GCD;          // Speaker must be driven at the fastest rate (Twice the fastest button).
        unsigned long int Button1Period = Button1Tick/GCD;
        unsigned long int Button2Period = Button2Tick/GCD;
        unsigned long int Button3Period = Button3Tick/GCD;
        unsigned long int KeyPadPeriod  = KeypadTick/GCD;

        // Declare an array of tasks
        static task task1, task2, task3, task4, task5;
        task *tasks[] = { &task1, &task2, &task3, &task4, &task5};
        const unsigned short numTasks = sizeof(tasks)/sizeof(task*);

        // Task 1
        task1.state = (int) SpeakerOff;     //Task initial state.
        task1.period = SpeakerPeriod;       //Task Period.
        task1.elapsedTime = SpeakerPeriod;  //Task current elapsed time.
        task1.TickFct = &SpeakerTask;       //Function pointer for the tick.

        // Task 2
        task2.state = (int) ButtonOff;     //Task initial state.
        task2.period = Button1Period;       //Task Period.
        task2.elapsedTime = Button1Period;  //Task current elapsed time.
        task2.TickFct = &Button1Task;       //Function pointer for the tick.

        // Task 3
        task3.state = (int) ButtonOff;     //Task initial state.
        task3.period = Button2Period;       //Task Period.
        task3.elapsedTime = Button2Period;  //Task current elapsed time.
        task3.TickFct = &Button2Task;       //Function pointer for the tick.

        // Task 4
        task4.state = (int) ButtonOff;     //Task initial state.
        task4.period = Button3Period;       //Task Period.
        task4.elapsedTime = Button3Period;  //Task current elapsed time.
        task4.TickFct = &Button3Task;       //Function pointer for the tick.

        // Task 5
        task5.state = (int) KeyPadOff;      //Task initial state.
        task5.period = KeyPadPeriod;        //Task Period.
        task5.elapsedTime = KeyPadPeriod;   //Task current elapsed time.
        task5.TickFct = &KeyPadTask;        //Function pointer for the tick.

        // Set the timer and turn it on
        TimerSet(GCD);
        TimerOn();

        // Run all tasks forever....
        unsigned short i; // Scheduler for-loop iterator
        while(1)
        {
                // Scheduler code
                for ( i = 0; i < numTasks; i++ )
                {
                        if (tasks[i]->period != 0)
                        {
                                // Task is ready to tick
                                if ( tasks[i]->elapsedTime == tasks[i]->period )
                                {
                                        // Setting next state for task
                                        tasks[i]->state = tasks[i]->TickFct(tasks[i]->state);
                                        // Reset the elapsed time for next tick.
                                        tasks[i]->elapsedTime = 0;
                                }
                                tasks[i]->elapsedTime += 1;
                        }
                }

                // Process_LCD_Task(1);
                while(!TimerFlag);
                TimerFlag = 0;
        }
}

int main(void)
{
   InitPorts();
   InitSpeaker();
   RunTaskScheduler();
}
```

```c
/*    adeja001_lab9_part5.c - 5-13-13
 *    Name & E-mail: Ariana Dejaco Adeja001@ucr.edu
 *    CS Login: adeja001
 *    Partner(s) Name & E-mail: Joshua DeForest-Williams jdefo002@ucr.edu
 *    Lab Section: 22
 *    Assignment: Lab # 9  Exercise # 5 Challenge
 *    Exercise Description:Using the ATmega32M-^Rs built in PWM functionality,
 *    create a piano with seven different tones that can be used to play a
 *    song.
 *
 *    I acknowledge all content contained herein, excluding template or example
 *    code, is my own original work.
 */

#include <avr/io.h>
#include <avr/sfr_defs.h>
#include <math.h>
#include "Bit.h"
#include "Buttons.h"
#include "KeyPad.h"
#include "Speaker_PartChallange.h"
#include "Timer0.h"
#include "Schedular.h"

/***********************************************************************/
/* Initialize the ports                                                */
/***********************************************************************/
void InitPorts()
{
        // Make PORTD7 as an output. This is connected to the speaker
        DDRD  = 0x80;
        PORTD = 0x7F;  // Do NOT use pull up resistor on
        // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
        // and PC[7:4] are outputs.
        DDRC  = 0xF0;
        PORTC = 0x0F;
        // Make Port A connected to the buttons. Make them all inputs
        DDRA = 0x00;
}

/***********************************************************************/
/* This function calculates the frequency used. The math is:           */
/* T = 1.0 / frequency - Converts to Period                            */
/* Ticks = T * 10,000 - Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
/* Toggle Rate = Ticks / 2.                                            */
/***********************************************************************/
unsigned long calculateFrequencyTick(double frequency)
{
        double Period_F = 1.0 / frequency;                                      // Period = 1 / Frequency
        double TicksPerSecond_F = Period_F * 10000.0;                   // Period * 10,000 ticks/second = Ticks.
        long   TicksPerSecond_L = (long) TicksPerSecond_F;              // Convert to a long from a float
        long   ToggleRate = TicksPerSecond_L / 2;                               // It takes two toggle to make 1 period
        return ToggleRate;
}

/***********************************************************************/
/* Initialize Task Scheduler                                           */
/***********************************************************************/
void RunTaskScheduler()
{
        unsigned long int GCD = 1;

        // Declare the local variables. The following are the ticks that the
        // button and keypads need to run in. Initially they are all zero (disabled)
        // They are added depending on which part of the lab we are running.
        unsigned long int Button1Tick = 0;  // Initially disabled
        unsigned long int Button2Tick = 0;  // Initially disabled
```

```c
        unsigned long int Button3Tick = 0;  // Initially disabled
        unsigned long int KeypadTick  = 0;  // Initially disabled

        ////////////////////////////////////////////////////////////////////////////////
        // Determine the real ticks that we need to run the button and keypad - If any..
        // Based on the lab part that we are running.
        ///////////////////////////////////////////////////////////////////////////////
#ifdef SPEAKER_PART1_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
#endif
#ifdef SPEAKER_PART2_H_
        Button1Tick = calculateFrequencyTick(Button1Frequency);
        Button2Tick = calculateFrequencyTick(Button2Frequency);
        Button3Tick = calculateFrequencyTick(Button3Frequency);
#endif
#ifdef SPEAKER_PART3_H_
        Button1Tick = 100;   // Good number for button sampling....
#endif
#ifdef SPEAKER_PART4_H_
        Button1Tick = 100;   // Good number for button sampling....
        Button2Tick = 10;    // Sounds good for increasing freq
        Button3Tick = 10;    // Sounds good for decreasing freq
#endif
#ifdef SPEAKER_PARTCHALLANGE_H_
        KeypadTick  = 100;   // Good number for button sampling....
#endif

        //Recalculate GCD periods for scheduler
        unsigned long int SpeakerPeriod = GCD;          // Speaker must be driven at the fastest rate (Twice the fastest button).
        unsigned long int Button1Period = Button1Tick/GCD;
        unsigned long int Button2Period = Button2Tick/GCD;
        unsigned long int Button3Period = Button3Tick/GCD;
        unsigned long int KeyPadPeriod  = KeypadTick/GCD;

        // Declare an array of tasks
        static task task1, task2, task3, task4, task5;
        task *tasks[] = { &task1, &task2, &task3, &task4, &task5};
        const unsigned short numTasks = sizeof(tasks)/sizeof(task*);

        // Task 1
        task1.state = (int) SpeakerOff;     //Task initial state.
        task1.period = SpeakerPeriod;       //Task Period.
        task1.elapsedTime = SpeakerPeriod;  //Task current elapsed time.
        task1.TickFct = &SpeakerTask;       //Function pointer for the tick.

        // Task 2
        task2.state = (int) ButtonOff;     //Task initial state.
        task2.period = Button1Period;       //Task Period.
        task2.elapsedTime = Button1Period;  //Task current elapsed time.
        task2.TickFct = &Button1Task;       //Function pointer for the tick.

        // Task 3
        task3.state = (int) ButtonOff;     //Task initial state.
        task3.period = Button2Period;       //Task Period.
        task3.elapsedTime = Button2Period;  //Task current elapsed time.
        task3.TickFct = &Button2Task;       //Function pointer for the tick.

        // Task 4
        task4.state = (int) ButtonOff;     //Task initial state.
        task4.period = Button3Period;       //Task Period.
        task4.elapsedTime = Button3Period;  //Task current elapsed time.
        task4.TickFct = &Button3Task;       //Function pointer for the tick.

        // Task 5
        task5.state = (int) KeyPadOff;      //Task initial state.
        task5.period = KeyPadPeriod;        //Task Period.
        task5.elapsedTime = KeyPadPeriod;   //Task current elapsed time.
        task5.TickFct = &KeyPadTask;        //Function pointer for the tick.

        // Set the timer and turn it on
        TimerSet(GCD);
        TimerOn();

        // Run all tasks forever....
        unsigned short i; // Scheduler for−loop iterator
        while(1)
        {
                // Scheduler code
                for ( i = 0; i < numTasks; i++ )
                {
                        if (tasks[i]−>period != 0)
                        {
                                // Task is ready to tick
                                if ( tasks[i]−>elapsedTime == tasks[i]−>period )
                                {
                                        // Setting next state for task
                                        tasks[i]−>state = tasks[i]−>TickFct(tasks[i]−>state);
                                        // Reset the elapsed time for next tick.
                                        tasks[i]−>elapsedTime = 0;
                                }
                                tasks[i]−>elapsedTime += 1;
                        }
                }

                // Process_LCD_Task(1);
                while(!TimerFlag);
                TimerFlag = 0;
        }
}

int main(void)
{
   InitPorts();
   InitSpeaker();
   RunTaskScheduler();
}
```