```
/*
 * 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;
```

```
144         }
145
146         // Action Code: When button is pressed toggle button value which will dr
    ive the speaker.
147         if (Button3State == ButtonOn)
148         {
149                 if (Button3ToggleValue == 1)
150                         Button3ToggleValue = 0;
151                 else
152                         Button3ToggleValue = 1;
153         }
154
155         // Return the current state
156         return (int) Button3State;
157 }
158
159 #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_ */
```

```
1
2  // Permission to copy is granted provided that this header remains intact.
3  // This software is provided with no warranties.
4
5  ////////////////////////////////////////////////////////////////////////////////
6
7  #ifndef SCHEDULER_H
8  #define SCHEDULER_H
9
10 ////////////////////////////////////////////////////////////////////////////////
11 //Functionality - finds the greatest common divisor of two values
12 //Parameter: Two long int's to find their GCD
13 //Returns: GCD else 0
14 unsigned long int findGCD(unsigned long int a, unsigned long int b)
15 {
16         unsigned long int c;
17         while(1)
18         {
19                 c = a % b;
20                 if( c == 0 ) { return b; }
21                 a = b;
22                 b = c;
23         }
24         return 0;
25 }
26
27 ////////////////////////////////////////////////////////////////////////////////
28 //Struct for Tasks represent a running process in our simple real-time operating
    system
29 typedef struct _task{
30         // Tasks should have members that include: state, period,
31         //a measurement of elapsed time, and a function pointer.
32         signed   char state;            //Task's current state
33         unsigned long period;           //Task period
34         unsigned long elapsedTime;      //Time elapsed since last task tick
35         int (*TickFct)(int);            //Task tick function
36 } task;
37
38
39
40 #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_ */
```

```
/*
 * 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 == Button
tonOff) && (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)) ||
```



```
                                ((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....
                        }
```

```
139                         break;
140
141                 // Placed for completeness. This should NEVER happen.
142                 default:
143                         SpeakerState = SpeakerOff;
144                         break;
145         }
146
147         // Action Code: The action code is to simply drive the speaker
148         DriveSpeaker();
149
150         // Return the current state
151         return (int) SpeakerState;
152 }
153
154 #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 != OtherK
ey))
                                SpeakerState = SpeakerOn;
                        break;

                case SpeakerOn:
                        if ((KeyPadState == KeyPadOff) || (KeyPadState == OtherK
ey))
                                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_ */
```

```
1
2  // Permission to copy is granted provided that this header remains intact.
3  // This software is provided with no warranties.
4
5  ////////////////////////////////////////////////////////////////////////////////
6
7  #ifndef TIMER_H
8  #define TIMER_H
9
10 #include <avr/interrupt.h>
11
12 volatile unsigned char TimerFlag = 0; // TimerISR() sets this to 1. C programmer
    should clear to 0.
13
14 // Internal variables for mapping AVR's ISR to our cleaner TimerISR model.
15 unsigned long _avr_timer_M = 1; // Start count from here, down to 0. Default 1ms
16 unsigned long _avr_timer_cntcurr = 0; // Current internal count of 1ms ticks
17
18 // Set TimerISR() to tick every M ms
19 void TimerSet(unsigned long M) {
20         _avr_timer_M = M;
21         _avr_timer_cntcurr = _avr_timer_M;
22 }
23
24 void TimerOn() {
25         // AVR timer/counter controller register TCCR0
26         TCCR0   = 0x0A; // bit3bit6=10: CTC mode (clear timer on compare)
27         // bit2bit1bit0=010: prescaler /8
28         // 00001010: 0x0A
29         // SO, 8 MHz clock or 8,000,000 /8 = 1,000,000 ticks/s
30         // Thus, TCNT0 register will count at 1,000,000 ticks/s
31
32         // AVR output compare register OCR0.
33         OCR0    = 100;  // Timer interrupt will be generated when TCNT0==OCR0
34         TIMSK   = 0x02; // bit1: OCIE0 -- enables compare match interrupt
35                                        // 1,000,000 / 100 = 0.00001
36
37         //Initialize avr counter
38         TCNT0 = 0;
39
40         // TimerISR will be called every _avr_timer_cntcurr milliseconds
41         _avr_timer_cntcurr = _avr_timer_M;
42
43         //Enable global interrupts
44         SREG |= 0x80;   // 0x80: 1000000
45 }
46
47 void TimerOff() {
48         TCCR0   = 0x00; // bit2bit1bit0=000: timer off
49 }
50
51 void TimerISR() {
52         TimerFlag = 1;
53 }
54
55 // In our approach, the C programmer does not touch this ISR, but rather TimerIS
   R()
56 ISR(TIMER0_COMP_vect)
57 {
58         // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn se
   ttings)
59         _avr_timer_cntcurr--;                   // Count down to 0 rather than u
   p to TOP
60         if (_avr_timer_cntcurr == 0) {  // results in a more efficient compare
61                 TimerISR();                             // Call the ISR that the
    user uses
62                 _avr_timer_cntcurr = _avr_timer_M;
63         }
64 }
65
66 #endif //TIMER_H
```

```
1  /*    adeja001_lab9_part1.c − 5−13−13
2  *    Name & E−mail: Ariana Dejaco adeja001@ucr.edu
3  *    CS Login: adeja001
4  *    Partner(s) Name & E−mail: Joshua DeForest−Williams jdefo002@ucr.edu
5  *    Lab Section: 22
6  *    Assignment: Lab # 9 Exercise # 1
7  *    Exercise Description: The goal of this part of the lab is to produce
8  *    a middle C on the speaker when a button is pressed and held.
9  *    I acknowledge all content contained herein, excluding template or example
10 *    code, is my own original work.
11 */
12 #include <avr/io.h>
13 #include <avr/sfr_defs.h>
14 #include <math.h>
15 #include "Bit.h"
16 #include "Buttons.h"
17 #include "KeyPad.h"
18 #include "Speaker_Part1.h"
19 #include "Timer0.h"
20 #include "Schedular.h"
21
22 /*******************************************************************************/
23 /* Initialize the ports                                                        */
24 /*******************************************************************************/
25 void InitPorts()
26 {
27         // Make PORTD7 as an output. This is connected to the speaker
28         DDRD  = 0x80;
29         PORTD = 0x7F;  // Do NOT use pull up resistor on
30         // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
31         // and PC[7:4] are outputs.
32         DDRC  = 0xF0;
33         PORTC = 0x0F;
34         // Make Port A connected to the buttons. Make them all inputs
35         DDRA = 0x00;
36 }
37
38 /*******************************************************************************/
39 /* This function calculates the frequency used. The math is:
   */
40 /* T = 1.0 / frequency − Converts to Period                                    */
41 /* Ticks = T * 10,000 − Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
42 /* Toggle Rate = Ticks / 2.                                                    */
43 /*******************************************************************************/
44 unsigned long calculateFrequencyTick(double frequency)
45 {
46         double Period_F = 1.0 / frequency;
   // Period = 1 / Frequency
47         double TicksPerSecond_F = Period_F * 10000.0;                  // Perio
   d * 10,000 ticks/second = Ticks.
48         long   TicksPerSecond_L = (long) TicksPerSecond_F;             // Conve
   rt to a long from a float
49         long   ToggleRate = TicksPerSecond_L / 2;
   // It takes two toggle to make 1 period
50         return ToggleRate;
51 }
52
53 /*******************************************************************************/
54 /* Initialize Task Scheduler                                                   */
55 /*******************************************************************************/
56 void RunTaskScheduler()
57 {
58         unsigned long int GCD = 1;
59
60         // Declare the local variables. The following are the ticks that the
61         // button and keypads need to run in. Initially they are all zero (disab
   led)
62         // They are added depending on which part of the lab we are running.
63         unsigned long int Button1Tick = 0;  // Initially disabled
64         unsigned long int Button2Tick = 0;  // Initially disabled
65         unsigned long int Button3Tick = 0;  // Initially disabled
66         unsigned long int KeypadTick  = 0;  // Initially disabled
67
```



```
68         ////////////////////////////////////////////////////////////////////////
   ////////
69         // Determine the real ticks that we need to run the button and keypad −
   If any..
70         // Based on the lab part that we are running.
71         ////////////////////////////////////////////////////////////////////////
   ////////
72 #ifdef SPEAKER_PART1_H_
73         Button1Tick = calculateFrequencyTick(Button1Frequency);
74 #endif
75 #ifdef SPEAKER_PART2_H_
76         Button1Tick = calculateFrequencyTick(Button1Frequency);
77         Button2Tick = calculateFrequencyTick(Button2Frequency);
78         Button3Tick = calculateFrequencyTick(Button3Frequency);
79 #endif
80 #ifdef SPEAKER_PART3_H_
81         Button1Tick = 100;   // Good number for button sampling....
82 #endif
83 #ifdef SPEAKER_PART4_H_
84         Button1Tick = 100;   // Good number for button sampling....
85         Button2Tick = 10;    // Sounds good for increasing freq
86         Button3Tick = 10;    // Sounds good for decreasing freq
87 #endif
88 #ifdef SPEAKER_PARTCHALLANGE_H_
89         KeypadTick  = 100;   // Good number for button sampling....
90 #endif
91
92         //Recalculate GCD periods for scheduler
93         unsigned long int SpeakerPeriod = GCD;          // Speaker must be drive
   n at the fastest rate (Twice the fastest button).
94         unsigned long int Button1Period = Button1Tick/GCD;
95         unsigned long int Button2Period = Button2Tick/GCD;
96         unsigned long int Button3Period = Button3Tick/GCD;
97         unsigned long int KeyPadPeriod  = KeypadTick/GCD;
98
99         // Declare an array of tasks
100        static task task1, task2, task3, task4, task5;
101        task *tasks[] = { &task1, &task2, &task3, &task4, &task5};
102        const unsigned short numTasks = sizeof(tasks)/sizeof(task*);
103
104        // Task 1
105        task1.state = (int) SpeakerOff;     //Task initial state.
106        task1.period = SpeakerPeriod;       //Task Period.
107        task1.elapsedTime = SpeakerPeriod;  //Task current elapsed time.
108        task1.TickFct = &SpeakerTask;       //Function pointer for the tick.
109
110        // Task 2
111        task2.state = (int) ButtonOff;     //Task initial state.
112        task2.period = Button1Period;       //Task Period.
113        task2.elapsedTime = Button1Period;  //Task current elapsed time.
114        task2.TickFct = &Button1Task;       //Function pointer for the tick.
115
116        // Task 3
117        task3.state = (int) ButtonOff;     //Task initial state.
118        task3.period = Button2Period;       //Task Period.
119        task3.elapsedTime = Button2Period;  //Task current elapsed time.
120        task3.TickFct = &Button2Task;       //Function pointer for the tick.
121
122        // Task 4
123        task4.state = (int) ButtonOff;     //Task initial state.
124        task4.period = Button3Period;       //Task Period.
125        task4.elapsedTime = Button3Period;  //Task current elapsed time.
126        task4.TickFct = &Button3Task;       //Function pointer for the tick.
127
128        // Task 5
129        task5.state = (int) KeyPadOff;      //Task initial state.
130        task5.period = KeyPadPeriod;        //Task Period.
131        task5.elapsedTime = KeyPadPeriod;   //Task current elapsed time.
132        task5.TickFct = &KeyPadTask;        //Function pointer for the tick.
133
134        // Set the timer and turn it on
135        TimerSet(GCD);
136        TimerOn();
```

```
137
138         // Run all tasks forever....
139         unsigned short i; // Scheduler for-loop iterator
140         while(1)
141         {
142                 // Scheduler code
143                 for ( i = 0; i < numTasks; i++ )
144                 {
145                         if (tasks[i]->period != 0)
146                         {
147                                 // Task is ready to tick
148                                 if ( tasks[i]->elapsedTime == tasks[i]->period )
149                                 {
150                                         // Setting next state for task
151                                         tasks[i]->state = tasks[i]->TickFct(task
    s[i]->state);
152                                         // Reset the elapsed time for next tick.
153                                         tasks[i]->elapsedTime = 0;
154                                 }
155                                 tasks[i]->elapsedTime += 1;
156                         }
157                 }
158
159                 // Process_LCD_Task(1);
160                 while(!TimerFlag);
161                 TimerFlag = 0;
162         }
163 }
164
165 int main(void)
166 {
167    InitPorts();
168    InitSpeaker();
169    RunTaskScheduler();
170 }
```

```
/*    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;                   // Perio
d * 10,000 ticks/second = Ticks.
        long   TicksPerSecond_L = (long) TicksPerSecond_F;              // Conve
rt 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 (disab
led)
        // 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 drive
n 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
```

```
137         TimerSet(GCD);
138         TimerOn();
139
140         // Run all tasks forever....
141         unsigned short i; // Scheduler for−loop iterator
142         while(1)
143         {
144                 // Scheduler code
145                 for ( i = 0; i < numTasks; i++ )
146                 {
147                         if (tasks[i]->period != 0)
148                         {
149                                 // Task is ready to tick
150                                 if ( tasks[i]->elapsedTime == tasks[i]->period )
151                                 {
152                                         // Setting next state for task
153                                         tasks[i]->state = tasks[i]->TickFct(task
    s[i]->state);
154                                         // Reset the elapsed time for next tick.
155                                         tasks[i]->elapsedTime = 0;
156                                 }
157                                 tasks[i]->elapsedTime += 1;
158                         }
159                 }
160
161                 // Process_LCD_Task(1);
162                 while(!TimerFlag);
163                 TimerFlag = 0;
164         }
165 }
166
167 int main(void)
168 {
169    InitPorts();
170    InitSpeaker();
171    RunTaskScheduler();
172 }
```

```
/*    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;                   // Perio
d * 10,000 ticks/second = Ticks.
        long   TicksPerSecond_L = (long) TicksPerSecond_F;              // Conve
rt 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 (disab
led)
        // 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 drive
n 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.
```

```
137
138         // Set the timer and turn it on
139         TimerSet(GCD);
140         TimerOn();
141
142         // Run all tasks forever....
143         unsigned short i; // Scheduler for−loop iterator
144         while(1)
145         {
146                 // Scheduler code
147                 for ( i = 0; i < numTasks; i++ )
148                 {
149                         if (tasks[i]−>period != 0)
150                         {
151                                 // Task is ready to tick
152                                 if ( tasks[i]−>elapsedTime == tasks[i]−>period )
153                                 {
154                                         // Setting next state for task
155                                         tasks[i]−>state = tasks[i]−>TickFct(task
    s[i]−>state);
156                                         // Reset the elapsed time for next tick.
157                                         tasks[i]−>elapsedTime = 0;
158                                 }
159                                 tasks[i]−>elapsedTime += 1;
160                         }
161                 }
162
163                 // Process_LCD_Task(1);
164                 while(!TimerFlag);
165                 TimerFlag = 0;
166         }
167 }
168
169 int main(void)
170 {
171    InitPorts();
172    InitSpeaker();
173    RunTaskScheduler();
174 }
```

```
1  /*    adeja001_lab9_part4.c − 5−13−13
2  *    Name & E−mail: Ariana Dejaco Adeja001@ucr.edu
3  *    CS Login: adeja001
4  *    Partner(s) Name & E−mail: Joshua DeForest−Williams jdefo002@ucr.edu
5  *    Lab Section: 22
6  *    Assignment: Lab # 9  Exercise # 4 Challenge
7  *    Exercise Description: Expand upon part 3 of the lab by adding three
8  *    additional buttons with the following functionality. One button will
9  *    toggle sound on/off. The other two buttons will be used to raise or
10 *    lower the tone. Holding the tone buttons down should raise/lower the
11 *    tone slowly (A note every second will be slow enough).
12 *
13 *    I acknowledge all content contained herein, excluding template or example
14 *    code, is my own original work.
15 */
16
17 #include <avr/io.h>
18 #include <avr/sfr_defs.h>
19 #include <math.h>
20 #include "Bit.h"
21 #include "Buttons.h"
22 #include "KeyPad.h"
23 #include "Speaker_Part4.h"
24 #include "Timer0.h"
25 #include "Schedular.h"
26
27 /*******************************************************************************/
28 /* Initialize the ports                                                        */
29 /*******************************************************************************/
30 void InitPorts()
31 {
32         // Make PORTD7 as an output. This is connected to the speaker
33         DDRD  = 0x80;
34         PORTD = 0x7F;  // Do NOT use pull up resistor on
35         // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
36         // and PC[7:4] are outputs.
37         DDRC  = 0xF0;
38         PORTC = 0x0F;
39         // Make Port A connected to the buttons. Make them all inputs
40         DDRA = 0x00;
41 }
42
43 /*******************************************************************************/
44 /* This function calculates the frequency used. The math is:
   */
45 /* T = 1.0 / frequency − Converts to Period                                    */
46 /* Ticks = T * 10,000 − Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
47 /* Toggle Rate = Ticks / 2.                                                    */
48 /*******************************************************************************/
49 unsigned long calculateFrequencyTick(double frequency)
50 {
51         double Period_F = 1.0 / frequency;
   // Period = 1 / Frequency
52         double TicksPerSecond_F = Period_F * 10000.0;                  // Perio
   d * 10,000 ticks/second = Ticks.
53         long   TicksPerSecond_L = (long) TicksPerSecond_F;             // Conve
   rt to a long from a float
54         long   ToggleRate = TicksPerSecond_L / 2;
   // It takes two toggle to make 1 period
55         return ToggleRate;
56 }
57
58 /*******************************************************************************/
59 /* Initialize Task Scheduler                                                   */
60 /*******************************************************************************/
61 void RunTaskScheduler()
62 {
63         unsigned long int GCD = 1;
64
65         // Declare the local variables. The following are the ticks that the
66         // button and keypads need to run in. Initially they are all zero (disab
   led)
67         // They are added depending on which part of the lab we are running.
```



```
68         unsigned long int Button1Tick = 0;  // Initially disabled
69         unsigned long int Button2Tick = 0;  // Initially disabled
70         unsigned long int Button3Tick = 0;  // Initially disabled
71         unsigned long int KeypadTick  = 0;  // Initially disabled
72
73         ////////////////////////////////////////////////////////////////////////
   ////////
74         // Determine the real ticks that we need to run the button and keypad −
   If any..
75         // Based on the lab part that we are running.
76         ////////////////////////////////////////////////////////////////////////
   ////////
77 #ifdef SPEAKER_PART1_H_
78         Button1Tick = calculateFrequencyTick(Button1Frequency);
79 #endif
80 #ifdef SPEAKER_PART2_H_
81         Button1Tick = calculateFrequencyTick(Button1Frequency);
82         Button2Tick = calculateFrequencyTick(Button2Frequency);
83         Button3Tick = calculateFrequencyTick(Button3Frequency);
84 #endif
85 #ifdef SPEAKER_PART3_H_
86         Button1Tick = 100;   // Good number for button sampling....
87 #endif
88 #ifdef SPEAKER_PART4_H_
89         Button1Tick = 100;   // Good number for button sampling....
90         Button2Tick = 10;    // Sounds good for increasing freq
91         Button3Tick = 10;    // Sounds good for decreasing freq
92 #endif
93 #ifdef SPEAKER_PARTCHALLANGE_H_
94         KeypadTick  = 100;   // Good number for button sampling....
95 #endif
96
97         //Recalculate GCD periods for scheduler
98         unsigned long int SpeakerPeriod = GCD;          // Speaker must be drive
   n at the fastest rate (Twice the fastest button).
99         unsigned long int Button1Period = Button1Tick/GCD;
100        unsigned long int Button2Period = Button2Tick/GCD;
101        unsigned long int Button3Period = Button3Tick/GCD;
102        unsigned long int KeyPadPeriod  = KeypadTick/GCD;
103
104        // Declare an array of tasks
105        static task task1, task2, task3, task4, task5;
106        task *tasks[] = { &task1, &task2, &task3, &task4, &task5};
107        const unsigned short numTasks = sizeof(tasks)/sizeof(task*);
108
109        // Task 1
110        task1.state = (int) SpeakerOff;     //Task initial state.
111        task1.period = SpeakerPeriod;       //Task Period.
112        task1.elapsedTime = SpeakerPeriod;  //Task current elapsed time.
113        task1.TickFct = &SpeakerTask;       //Function pointer for the tick.
114
115        // Task 2
116        task2.state = (int) ButtonOff;     //Task initial state.
117        task2.period = Button1Period;       //Task Period.
118        task2.elapsedTime = Button1Period;  //Task current elapsed time.
119        task2.TickFct = &Button1Task;       //Function pointer for the tick.
120
121        // Task 3
122        task3.state = (int) ButtonOff;     //Task initial state.
123        task3.period = Button2Period;       //Task Period.
124        task3.elapsedTime = Button2Period;  //Task current elapsed time.
125        task3.TickFct = &Button2Task;       //Function pointer for the tick.
126
127        // Task 4
128        task4.state = (int) ButtonOff;     //Task initial state.
129        task4.period = Button3Period;       //Task Period.
130        task4.elapsedTime = Button3Period;  //Task current elapsed time.
131        task4.TickFct = &Button3Task;       //Function pointer for the tick.
132
133        // Task 5
134        task5.state = (int) KeyPadOff;      //Task initial state.
135        task5.period = KeyPadPeriod;        //Task Period.
136        task5.elapsedTime = KeyPadPeriod;   //Task current elapsed time.
```

```
137         task5.TickFct = &KeyPadTask;        //Function pointer for the tick.
138
139         // Set the timer and turn it on
140         TimerSet(GCD);
141         TimerOn();
142
143         // Run all tasks forever....
144         unsigned short i; // Scheduler for−loop iterator
145         while(1)
146         {
147                 // Scheduler code
148                 for ( i = 0; i < numTasks; i++ )
149                 {
150                         if (tasks[i]−>period != 0)
151                         {
152                                 // Task is ready to tick
153                                 if ( tasks[i]−>elapsedTime == tasks[i]−>period )
154                                 {
155                                         // Setting next state for task
156                                         tasks[i]−>state = tasks[i]−>TickFct(task
    s[i]−>state);
157                                         // Reset the elapsed time for next tick.
158                                         tasks[i]−>elapsedTime = 0;
159                                 }
160                                 tasks[i]−>elapsedTime += 1;
161                         }
162                 }
163
164                 // Process_LCD_Task(1);
165                 while(!TimerFlag);
166                 TimerFlag = 0;
167         }
168 }
169
170 int main(void)
171 {
172    InitPorts();
173    InitSpeaker();
174    RunTaskScheduler();
175 }
```

```
/*    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;                   // Perio
d * 10,000 ticks/second = Ticks.
        long   TicksPerSecond_L = (long) TicksPerSecond_F;              // Conve
rt 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 (disab
led)
        // 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 drive
n 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.
```

```
137         // Set the timer and turn it on
138         TimerSet(GCD);
139         TimerOn();
140
141         // Run all tasks forever....
142         unsigned short i; // Scheduler for−loop iterator
143         while(1)
144         {
145                 // Scheduler code
146                 for ( i = 0; i < numTasks; i++ )
147                 {
148                         if (tasks[i]−>period != 0)
149                         {
150                                 // Task is ready to tick
151                                 if ( tasks[i]−>elapsedTime == tasks[i]−>period )
152                                 {
153                                         // Setting next state for task
154                                         tasks[i]−>state = tasks[i]−>TickFct(task
    s[i]−>state);
155                                         // Reset the elapsed time for next tick.
156                                         tasks[i]−>elapsedTime = 0;
157                                 }
158                                 tasks[i]−>elapsedTime += 1;
159                         }
160                 }
161
162                 // Process_LCD_Task(1);
163                 while(!TimerFlag);
164                 TimerFlag = 0;
165         }
166 }
167
168 int main(void)
169 {
170    InitPorts();
171    InitSpeaker();
172    RunTaskScheduler();
173 }
```