```
1   /*
2    * Buttons.h
3    *
4    * Created: 5/13/2013 7:24:49 AM
5    *  Author: Ariana DeJaco
6    */
7
8
9   #ifndef BUTTONS_H_
10  #define BUTTONS_H_
11
12  // The following states are for the buttons
13  enum Button_States {ButtonOff, ButtonOn} Button1State, Button2State, Button3Stat
    e;
14
15  // The following toggle values are used to drive the speaker
16  unsigned char Button1ToggleValue, Button2ToggleValue, Button3ToggleValue;
17
18  // Define the frequencies for each button
19  #define Button1Frequency 261.63
20  #define Button2Frequency 293.66
21  #define Button3Frequency 329.63
22
23  /**************************************************************************/
24  /* This function represents a Tick on the Button1 state machine.        */
25  /**************************************************************************/
26  int Button1Task(int currentState)
27  {
28          unsigned char button1Value = GetBit(PINA, 0);
29
30          Button1State = (enum Button_States) currentState;
31
32          // State Transitions
33          switch (Button1State)
34          {
35                  case ButtonOff:
36                          if (button1Value == 0)
37                                  Button1State = ButtonOn;
38                          else
39                                  Button1State = ButtonOff;
40                          break;
41
42                  case ButtonOn:
43                          if (button1Value != 0)
44                                  Button1State = ButtonOff;
45                          else
46                                  Button1State = ButtonOn;
47                          break;
48
49                  // Placed for completeness. This should NEVER happen.
50                  default:
51                          Button1State = ButtonOff;
52                          break;
53          }
54
55          // Action Code: When button is pressed toggle button value which will dr
    ive the speaker.
56          if (Button1State == ButtonOn)
57          {
58                  if (Button1ToggleValue == 1)
59                          Button1ToggleValue = 0;
60                  else
61                          Button1ToggleValue = 1;
62          }
63
64          // Return the current state
65          return (int) Button1State;
66  }
67
68  /**************************************************************************/
69  /* This function represents a Tick on the Button2 state machine.        */
70  /**************************************************************************/
71  int Button2Task(int currentState)
```

```
72  {
73          unsigned char button2Value = GetBit(PINA, 1);
74
75          Button2State = (enum Button_States) currentState;
76
77          // State Transitions
78          switch (Button2State)
79          {
80                  case ButtonOff:
81                          if (button2Value != 0)
82                                  Button2State = ButtonOn;
83                          else
84                                  Button2State = ButtonOff;
85                          break;
86
87                  case ButtonOn:
88                          if (button2Value == 0)
89                                  Button2State = ButtonOff;
90                          else
91                                  Button2State = ButtonOn;
92                          break;
93
94                  // Placed for completeness. This should NEVER happen.
95                  default:
96                          Button2State = ButtonOff;
97                          break;
98          }
99
100         // Action Code: When button is pressed toggle button value which will dr
    ive the speaker.
101         if (Button2State == ButtonOn)
102         {
103                 if (Button2ToggleValue == 1)
104                         Button2ToggleValue = 0;
105                 else
106                         Button2ToggleValue = 1;
107         }
108
109         // Return the current state
110         return (int) Button2State;
111 }
112
113
114 /**************************************************************************/
115 /* This function represents a Tick on the Button3 state machine.        */
116 /**************************************************************************/
117 int Button3Task(int currentState)
118 {
119         unsigned char button3Value = GetBit(PINA, 2);
120
121         Button3State = (enum Button_States) currentState;
122
123         // State Transitions
124         switch (Button3State)
125         {
126                 case ButtonOff:
127                         if (button3Value != 0)
128                                 Button3State = ButtonOn;
129                         else
130                                 Button3State = ButtonOff;
131                         break;
132
133                 case ButtonOn:
134                         if (button3Value == 0)
135                                 Button3State = ButtonOff;
136                         else
137                                 Button3State = ButtonOn;
138                         break;
139
140                 // Placed for completeness. This should NEVER happen.
141                 default:
142                         Button3State = ButtonOff;
143                         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_ */
```

```c
1   /*
2    * KeyPad.h
3    *
4    * Created: 5/13/2013 6:50:46 AM
5    *  Author: Ariana DeJaco
6    */
7
8   #ifndef KEYPAD_H_
9   #define KEYPAD_H_
10
11  // The following states are for the buttons
12  enum KeyPad_States {KeyPadOff, Key1, Key2, Key3, Key4, Key5, Key6, Key7, OtherKey} KeyPadState;
13
14
15  #define KEY1_FREQUENCY  261.63
16  #define KEY2_FREQUENCY  293.66
17  #define KEY3_FREQUENCY  329.63
18  #define KEY4_FREQUENCY  349.23
19  #define KEY5_FREQUENCY  392.00
20  #define KEY6_FREQUENCY  440.00
21  #define KEY7_FREQUENCY  493.88
22
23  double KeyPadFrequency;
24
25  /**************************************************************************/
26  /* Returns '\0' if no key pressed, else returns char '1', '2', ...       */
27  /* If multiple keys pressed, returns leftmost-topmost one                */
28  /* Keypad must be connected to port C                                    */
29  /* Keypad arrangement                                                    */
30  /*        PC4 PC5 PC6 PC7                                                 */
31  /*   col  1   2   3   4                                                   */
32  /* row                                                                   */
33  /* PC0 1   1 | 2 | 3 | A                                                  */
34  /* PC1 2   4 | 5 | 6 | B                                                  */
35  /* PC2 3   7 | 8 | 9 | C                                                  */
36  /* PC3 4   * | 0 | # | D                                                  */
37  /**************************************************************************/
38
39  unsigned char GetKeypadKey()
40  {
41          PORTC = 0xEF; // Enable col 4 with 0, disable others with 1M-^Rs
42          asm("nop"); // add a delay to allow PORTC to stabilize before checking
43          if (GetBit(PINC,0)==0) { return('1'); }
44          if (GetBit(PINC,1)==0) { return('4'); }
45          if (GetBit(PINC,2)==0) { return('7'); }
46          if (GetBit(PINC,3)==0) { return('*'); }
47
48          // Check keys in col 2
49          PORTC = 0xDF; // Enable col 5 with 0, disable others with 1M-^Rs
50          asm("nop"); // add a delay to allow PORTC to stabilize before checking
51          if (GetBit(PINC,0)==0) { return('2'); }
52          if (GetBit(PINC,1)==0) { return('5'); }
53          if (GetBit(PINC,2)==0) { return('8'); }
54          if (GetBit(PINC,3)==0) { return('0'); }
55
56          // Check keys in col 3
57          PORTC = 0xBF; // Enable col 6 with 0, disable others with 1M-^Rs
58          asm("nop"); // add a delay to allow PORTC to stabilize before checking
59          if (GetBit(PINC,0)==0) { return('3'); }
60          if (GetBit(PINC,1)==0) { return('6'); }
61          if (GetBit(PINC,2)==0) { return('9'); }
62          if (GetBit(PINC,3)==0) { return('#'); }
63
64          // Check keys in col 4
65          PORTC = 0x7F;
66          asm("nop"); // add a delay to allow PORTC to stabilize before checking
67          if (GetBit(PINC,0)==0) { return('A'); }
68          if (GetBit(PINC,1)==0) { return('B'); }
69          if (GetBit(PINC,2)==0) { return('C'); }
70          if (GetBit(PINC,3)==0) { return('D'); }
71
72          return('\0'); // default value
```

```c
73  }
74
75  /**************************************************************************/
76  /* This function represents a Tick on the Button1 state machine.        */
77  /**************************************************************************/
78  int KeyPadTask(int currentState)
79  {
80          // Switching states ONLY depends on the keypad value and not the current state.
81          // The state is just the keypad value.
82          switch (GetKeypadKey())
83          {
84                  case '\0': KeyPadState = KeyPadOff; break;
85                  case '1':  KeyPadState = Key1;       break;
86                  case '2':  KeyPadState = Key2;                break;
87                  case '3':  KeyPadState = Key3;                break;
88                  case '4':  KeyPadState = Key4;                break;
89                  case '5':  KeyPadState = Key5;                break;
90                  case '6':  KeyPadState = Key6;                break;
91                  case '7':  KeyPadState = Key7;                break;
92                  default:   KeyPadState = OtherKey;            break;
93          }
94
95          // Action Code
96          switch (KeyPadState)
97          {
98                  case KeyPadOff: KeyPadFrequency = 0.0;
99          break;
                    case Key1:              KeyPadFrequency = KEY1_FREQUENCY;
100         break;
                    case Key2:              KeyPadFrequency = KEY2_FREQUENCY;
101         break;
                    case Key3:              KeyPadFrequency = KEY3_FREQUENCY;
102         break;
                    case Key4:              KeyPadFrequency = KEY4_FREQUENCY;
103         break;
                    case Key5:              KeyPadFrequency = KEY5_FREQUENCY;
104         break;
                    case Key6:              KeyPadFrequency = KEY6_FREQUENCY;
105         break;
                    case Key7:              KeyPadFrequency = KEY7_FREQUENCY;
106         break;
                    case OtherKey:  KeyPadFrequency = 0.0;
107         break;
                };
108
109         // Return the current state
110         return KeyPadState;
111 }
112
113 #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
29      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
```

```
1   /*
2    * Speaker_Part1.h
3    *
4    * Created: 5/13/2013 1:00:48 PM
5    *  Author: Ariana DeJaco
6    */
7
8
9   #ifndef SPEAKER_PART1_H_
10  #define SPEAKER_PART1_H_
11
12  // The following states are for the speaker
13  enum    Speaker_States {SpeakerOff, SpeakerOn} SpeakerState;
14
15  /***************************************************************************/
16  /* This function will drive the speaker if the Speaker toggle value      */
17  /* is 0 then D7 is zero otherwise D7 is driven high.                     */
18  /***************************************************************************/
19  void DriveSpeaker()
20  {
21          // Pulse D7 to drive the speaker
22          if (SpeakerState == SpeakerOn)
23          {
24                  if (Button1ToggleValue == 0)
25                          PORTD &= 0x7F;                  // Turn OFF D7
26                  else
27                          PORTD |= 0x80;                  // Turn ON D7
28          }
29          else
30          {
31                  PORTD &= 0x7F;                          // Turn OFF D7
32          }
33  }
34
35  /***************************************************************************/
36  /*                                                                       */
37  /***************************************************************************/
38  void InitSpeaker()
39  {
40          SpeakerState = SpeakerOff;
41  }
42
43  /***************************************************************************/
44  /* This function represents a Tick on the Button1 state machine.         */
45  /***************************************************************************/
46  int SpeakerTask(int currentState)
47  {
48          SpeakerState = (enum Speaker_States) currentState;
49
50          // State Transitions
51          switch (SpeakerState)
52          {
53                  case SpeakerOff:
54                          if (Button1State == ButtonOn)
55                                  SpeakerState = SpeakerOn;
56                          break;
57
58                  case SpeakerOn:
59                          if (Button1State == ButtonOff)
60                                  SpeakerState = SpeakerOff;
61                          break;
62
63                  // Placed for completeness. This should NEVER happen.
64                  default:
65                          SpeakerState = SpeakerOff;
66                          break;
67          }
68
69          // Action Code: The action code is to simply drive the speaker
70          DriveSpeaker();
71
72          // Return the current state
73          return (int) SpeakerState;
```

```
74  }
75
76  #endif /* SPEAKER_PART1_H_ */
```

```
1  /*
2   * Speaker_Part2.h
3   *
4   * Created: 5/13/2013 1:03:49 PM
5   *  Author: Ariana DeJaco
6   */
7
8  #ifndef SPEAKER_PART2_H_
9  #define SPEAKER_PART2_H_
10
11 // The following states are for the speaker
12 enum   Speaker_States {SpeakerOff, SpeakerOn} SpeakerState;
13
14 /***************************************************************************/
15 /* This function will drive the speaker if the Speaker toggle value     */
16 /* is 0 then D7 is zero otherwise D7 is driven high.                    */
17 /***************************************************************************/
18 void DriveSpeaker()
19 {
20         unsigned char SpeakerToggleValue;
21         if (SpeakerState == SpeakerOn)
22         {
23                 if (Button1State == ButtonOn)
24                         SpeakerToggleValue = Button1ToggleValue;
25                 else if (Button2State == ButtonOn)
26                         SpeakerToggleValue = Button2ToggleValue;
27                 else
28                         SpeakerToggleValue = Button3ToggleValue;
29
30                 if (SpeakerToggleValue == 0)
31                         PORTD &= 0x7F;                  // Turn OFF D7
32                 else
33                         PORTD |= 0x80;                  // Turn ON D7
34         }
35         else
36         {
37                 PORTD &= 0x7F;                          // Turn OFF D7
38         }
39 }
40 /***************************************************************************/
41 /*                                                                       */
42 /*                                                                       */
43 /***************************************************************************/
44 void InitSpeaker()
45 {
46         SpeakerState = SpeakerOff;
47 }
48
49 /***************************************************************************/
50 /* This function represents a Tick on the Button1 state machine.        */
51 /***************************************************************************/
52 int SpeakerTask(int currentState)
53 {
54         SpeakerState = (enum Speaker_States) currentState;
55
56         // State Transitions
57         switch (SpeakerState)
58         {
59                 case SpeakerOff:
60                         if (((Button1State == ButtonOn)  && (Button2State == But
   tonOff) && (Button3State == ButtonOff)) ||
61                                 ((Button1State == ButtonOff) && (Button2State ==
    ButtonOn)  && (Button3State == ButtonOff)) ||
62                                 ((Button1State == ButtonOff) && (Button2State ==
    ButtonOff) && (Button3State == ButtonOn)))
63                                 SpeakerState = SpeakerOn;
64                         break;
65
66                 case SpeakerOn:
67                         if (! (((Button1State == ButtonOn)  && (Button2State ==
   ButtonOff) && (Button3State == ButtonOff)) ||
68                                 ((Button1State == ButtonOff) && (Button2State
    == ButtonOn)  && (Button3State == ButtonOff)) ||
```

```
69                                 ((Button1State == ButtonOff) && (Button2State
    == ButtonOff) && (Button3State == ButtonOn))))
70                                 SpeakerState = SpeakerOff;
71                         break;
72
73                 // Placed for completeness. This should NEVER happen.
74                 default:
75                         SpeakerState = SpeakerOff;
76                         break;
77         }
78
79         // Action Code: The action code is to simply drive the speaker
80         DriveSpeaker();
81
82         // Return the current state
83         return (int) SpeakerState;
84 }
85
86 #endif /* SPEAKER_PART2_H_ */
```

```
1   /*
2    * Speaker_Part3.h
3    *
4    * Created: 5/13/2013 1:08:28 PM
5    *  Author: Ariana DeJaco
6    */
7
8   #ifndef SPEAKER_PART3_H_
9   #define SPEAKER_PART3_H_
10
11  // The following states are for the speaker
12  enum   Speaker_States {SpeakerOff, SpeakerOn} SpeakerState;
13
14  /***********************************************************************/
15  /* This code is for PART 3 of the lab and is used to configures the D7  */
16  /* to toggle automatically using Timer2 of ATMega32.                   */
17  /* Timer2 Page 125: WGM21 = 1, WGM20 = 0 is CTC mode.                  */
18  /*           CTC mode is clear timer on compare match. (page 119)      */
19  /* Timer2 Page 126: COM21 = 0, COM20 = 1 is Toggle on compare.        */
20  /* Timer2 Page 127: CS22 = 1, CS21 = 0, CS20 = 0 -> Clk / 64.         */
21  /***********************************************************************/
22  void InitPWM()
23  {
24          TCCR2 = (1 << WGM21) | (1 << COM20) | (1 << CS22) ;
25  }
26
27  /***********************************************************************/
28  /* Sets the frequency of the Toggle Pin.                               */
29  /* The correct equation is on of page 120. If you solve for OCRn you   */
30  /* get the equation listed below for converting freq to OCR2.          */
31  /***********************************************************************/
32  void set_PWM (double frequency)
33  {
34          if (frequency < 1)
35          OCR2 = 0;
36          else
37          OCR2 = (unsigned char) ((8000000.0 / (128.0 * frequency)) - 1.0);
38  }
39
40  /***********************************************************************/
41  /* This function will drive the speaker if the Speaker toggle value    */
42  /* is 0 then D7 is zero otherwise D7 is driven high.                   */
43  /***********************************************************************/
44  void DriveSpeaker()
45  {
46          if (SpeakerState == SpeakerOn)
47          {
48                  // Select the frequency based on the button pressed
49                  if (Button1State == ButtonOn)
50                  {
51                          set_PWM(Button1Frequency);
52                  }
53                  else
54                  {
55                          set_PWM(0.0);
56                  }
57          }
58          else
59          {
60                  set_PWM(0.0);
61          }
62  }
63
64  /***********************************************************************/
65  /*                                                                     */
66  /***********************************************************************/
67  void InitSpeaker()
68  {
69          SpeakerState = SpeakerOff;
70          InitPWM();
71  }
72
73
```

```
74  /***********************************************************************/
75  /* This function represents a Tick on the Button1 state machine.       */
76  /***********************************************************************/
77  int SpeakerTask(int currentState)
78  {
79          SpeakerState = (enum Speaker_States) currentState;
80
81          // State Transitions
82          switch (SpeakerState)
83          {
84                  case SpeakerOff:
85                          if (Button1State == ButtonOn)
86                                  SpeakerState = SpeakerOn;
87                          break;
88
89                  case SpeakerOn:
90                          if (Button1State == ButtonOff)
91                                  SpeakerState = SpeakerOff;
92                          break;
93
94                  // Placed for completeness. This should NEVER happen.
95                  default:
96                          SpeakerState = SpeakerOff;
97                          break;
98          }
99
100         // Action Code: The action code is to simply drive the speaker
101         DriveSpeaker();
102
103         // Return the current state
104         return (int) SpeakerState;
105 }
106
107 #endif /* SPEAKER_PART3_H_ */
```

```
1   /*
2    * Speaker_Part4.h
3    *
4    * Created: 5/13/2013 1:19:05 PM
5    *  Author: Ariana DeJaco
6    */
7
8
9   #ifndef SPEAKER_PART4_H_
10  #define SPEAKER_PART4_H_
11
12  #define DEBOUNCE_COUNT_LIMIT 200
13
14  // The following states are for the speaker
15  enum   Speaker_States {SpeakerOff, SpeakerOn, SpeakerOnWaitForRelease, SpeakerO
    ffWaitForRelease} SpeakerState;
16  double  SpeakerFrequency;
17  unsigned char   DebounceCounter;
18  int     ToneTimer;
19
20  /***********************************************************************/
21  /* This code is for PART 3 of the lab and is used to configures the D7  */
22  /* to toggle automatically using Timer2 of ATMega32.                    */
23  /* Timer2 Page 125: WGM21 = 1, WGM20 = 0 is CTC mode.                   */
24  /*          CTC mode is clear timer on compare match. (page 119)        */
25  /* Timer2 Page 126: COM21 = 0, COM20 = 1 is Toggle on compare.          */
26  /* Timer2 Page 127: CS22 = 1, CS21 = 0, CS20 = 0 -> Clk / 64.           */
27  /***********************************************************************/
28  void InitPWM()
29  {
30          TCCR2 = (1 << WGM21) | (1 << COM20) | (1 << CS22) ;
31  }
32
33  /***********************************************************************/
34  /* Sets the frequency of the Toggle Pin.                                */
35  /* The correct equation is on of page 120. If you solve for OCRn you    */
36  /* get the equation listed below for converting freq to OCR2.           */
37  /***********************************************************************/
38  void set_PWM (double frequency)
39  {
40          if (frequency < 1)
41          OCR2 = 0;
42          else
43          OCR2 = (unsigned char) ((8000000.0 / (128.0 * frequency)) - 1.0);
44  }
45  /***********************************************************************/
46  /* This function will drive the speaker if the Speaker toggle value     */
47  /* is 0 then D7 is zero otherwise D7 is driven high.                    */
48  /***********************************************************************/
49
50  void DriveSpeaker()
51  {
52          if ((SpeakerState == SpeakerOn) || (SpeakerState == SpeakerOnWaitForRele
    ase))
53          {
54                  // Select the frequency based on the button pressed
55                  if ((Button2State == ButtonOn) && (Button3State == ButtonOff))
56                  {
57                          SpeakerFrequency += 0.001;
58                  }
59                  else if ((Button2State == ButtonOff) && (Button3State == ButtonO
    n))
60                  {
61                          SpeakerFrequency -= 0.001;
62                  }
63                  else
64                          SpeakerFrequency = SpeakerFrequency; // Don't change the
    frequency....
65
66                  // Drive the speaker with this frequency
67                  set_PWM(SpeakerFrequency);
68          }
69          else
```

```
70          {
71                  set_PWM(0.0);
72          }
73  }
74
75  /***********************************************************************/
76  /*                                                                     */
77  /***********************************************************************/
78  void InitSpeaker()
79  {
80          SpeakerState = SpeakerOff;
81          DebounceCounter  = 0;
82          SpeakerFrequency = 349.23;
83          InitPWM();
84  }
85
86
87  /***********************************************************************/
88  /* This function represents a Tick on the Button1 state machine.       */
89  /***********************************************************************/
90  int SpeakerTask(int currentState)
91  {
92          SpeakerState = (enum Speaker_States) currentState;
93
94          // State Transitions
95          switch (SpeakerState)
96          {
97                  case SpeakerOff:
98                          if (Button1State == ButtonOn)
99                          {
100                                 SpeakerState = SpeakerOnWaitForRelease;
101                                 DebounceCounter = 0;
102                         }
103                         break;
104
105                 case SpeakerOn:
106                         if (Button1State == ButtonOn)
107                         {
108                                 SpeakerState = SpeakerOffWaitForRelease;
109                                 DebounceCounter = 0;
110                         }
111                         break;
112
113                 case SpeakerOffWaitForRelease:
114                         if (DebounceCounter < DEBOUNCE_COUNT_LIMIT)
115                                 DebounceCounter++;  // Wait until we hit DEBOUNC
    E_COUNT_LIMIT
116                         else
117                         {
118                                 if (Button1State == ButtonOff)
119                                 {
120                                         SpeakerState = SpeakerOff;
121                                 }
122                                 else
123                                         DebounceCounter = 0;    // Wait some mor
    e...
124                         }
125                         break;
126
127                 case SpeakerOnWaitForRelease:
128                         if (DebounceCounter < DEBOUNCE_COUNT_LIMIT)
129                                 DebounceCounter++;  // Wait until we hit DEBOUNC
    E_COUNT_LIMIT
130                         else
131                         {
132                                 if (Button1State == ButtonOff)
133                                 {
134                                         SpeakerState = SpeakerOn;
135                                 }
136                                 else
137                                         DebounceCounter = 0;    // Wait some mor
    e....
138                         }
```

```
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_ */
```

```
1   /*
2    * Speaker_PartChallange.h
3    *
4    * Created: 5/13/2013 1:55:57 PM
5    *  Author: Ariana DeJaco
6    */
7
8   #ifndef SPEAKER_PARTCHALLANGE_H_
9   #define SPEAKER_PARTCHALLANGE_H_
10
11  // The following states are for the speaker
12  enum Speaker_States {SpeakerOff, SpeakerOn} SpeakerState;
13
14  /***********************************************************************/
15  /* This code is for PART 3 of the lab and is used to configures the D7  */
16  /* to toggle automatically using Timer2 of ATMega32.                   */
17  /* Timer2 Page 125: WGM21 = 1, WGM20 = 0 is CTC mode.                  */
18  /*           CTC mode is clear timer on compare match. (page 119)      */
19  /* Timer2 Page 126: COM21 = 0, COM20 = 1 is Toggle on compare.        */
20  /* Timer2 Page 127: CS22 = 1, CS21 = 0, CS20 = 0 -> Clk / 64.         */
21  /***********************************************************************/
22  void InitPWM()
23  {
24          TCCR2 = (1 << WGM21) | (1 << COM20) | (1 << CS22) ;
25  }
26
27  /***********************************************************************/
28  /* Sets the frequency of the Toggle Pin.                              */
29  /* The correct equation is on of page 120. If you solve for OCRn you   */
30  /* get the equation listed below for converting freq to OCR2.         */
31  /***********************************************************************/
32  void set_PWM (double frequency)
33  {
34          if (frequency < 1)
35                  OCR2 = 0;
36          else
37                  OCR2 = (unsigned char) ((8000000.0 / (128.0 * frequency)) - 1.0)
    ;
38  }
39
40  /***********************************************************************/
41  /* This function will drive the speaker if the Speaker toggle value    */
42  /* is 0 then D7 is zero otherwise D7 is driven high.                  */
43  /***********************************************************************/
44  void DriveSpeaker()
45  {
46          if (SpeakerState == SpeakerOn)
47          {
48                  set_PWM(KeyPadFrequency);
49          }
50          else
51          {
52                  set_PWM(0.0);
53          }
54  }
55
56  /***********************************************************************/
57  /*                                                                    */
58  /***********************************************************************/
59  void InitSpeaker()
60  {
61          SpeakerState = SpeakerOff;
62          InitPWM();
63  }
64
65  /***********************************************************************/
66  /* This function represents a Tick on the Button1 state machine.      */
67  /***********************************************************************/
68  int SpeakerTask(int currentState)
69  {
70          SpeakerState = (enum Speaker_States) currentState;
71
72          // State Transitions
```

```
73          switch (SpeakerState)
74          {
75                  case SpeakerOff:
76                          if ((KeyPadState != KeyPadOff) && (KeyPadState != OtherK
    ey))
77                                  SpeakerState = SpeakerOn;
78                          break;
79
80                  case SpeakerOn:
81                          if ((KeyPadState == KeyPadOff) || (KeyPadState == OtherK
    ey))
82                                  SpeakerState = SpeakerOff;
83                          break;
84
85                  // Placed for completeness. This should NEVER happen.
86                  default:
87                          SpeakerState = SpeakerOff;
88                          break;
89          }
90
91          // Action Code: The action code is to simply drive the speaker
92          DriveSpeaker();
93
94          // Return the current state
95          return (int) SpeakerState;
96  }
97
98  #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
```

```c
1   /*    jdefo002_lab9_part1.c - 5-13-13
2    *    Name & E-mail: Joshua DeForest-Williams jdefo002@ucr.edu
3    *    CS Login: jdefo002
4    *    Partner(s) Name & E-mail: Ariana Dejaco adeja001@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   */
41   /* T = 1.0 / frequency - Converts to Period                            */
42   /* Ticks = T * 10,000 - Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
43   /* Toggle Rate = Ticks / 2.                                            */
44   /***********************************************************************/
45   unsigned long calculateFrequencyTick(double frequency)
46   {
47       double Period_F = 1.0 / frequency;
48   // Period = 1 / Frequency
49       double TicksPerSecond_F = Period_F * 10000.0;                  // Perio
50   d * 10,000 ticks/second = Ticks.
51       long   TicksPerSecond_L = (long) TicksPerSecond_F;        // Convert to a
52   long from a float
53       long   ToggleRate = TicksPerSecond_L / 2;                  // It ta
54   kes 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 (disabled)
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       ////////////////////////////////////////////////////////////////////
```

Note: Line numbers in original: 46→line 46, 47→Period_F; 47→TicksPerSecond; etc.

```c
        ///
69      // Determine the real ticks that we need to run the button and keypad - If an
y..
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
```

```c
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(tasks[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  }
```

```c
1   /*    jdefo002_lab9_part2.c - 5-13-13
2    *    Name & E-mail: Joshua DeForest-Williams jdefo002@ucr.edu
3    *    CS Login: jdefo002
4    *    Partner(s) Name & E-mail: Ariana Dejaco adeja001@ucr.edu
5    *    Lab Section: 22
6    *    Assignment: Lab # 9 Exercise # 2
7    *    Exercise Description: For this part of the lab, create a system that
8    *    has three button inputs and, depending on which button is pressed,
9    *    the speaker will output either a middle C, D, or E.
10   *
11   *    I acknowledge all content contained herein, excluding template or example
12   *    code, is my own original work.
13   */
14   #include <avr/io.h>
15   #include <avr/sfr_defs.h>
16   #include <math.h>
17   #include "Bit.h"
18   #include "Buttons.h"
19   #include "KeyPad.h"
20   #include "Speaker_Part2.h"
21   #include "Timer0.h"
22   #include "Schedular.h"
23
24   /*************************************************************************/
25   /* Initialize the ports                                                  */
26   /*************************************************************************/
27   void InitPorts()
28   {
29       // Make PORTD7 as an output. This is connected to the speaker
30       DDRD  = 0x80;
31       PORTD = 0x7F;  // Do NOT use pull up resistor on
32       // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
33       // and PC[7:4] are outputs.
34       DDRC  = 0xF0;
35       PORTC = 0x0F;
36       // Make Port A connected to the buttons. Make them all inputs
37       DDRA = 0x00;
38   }
39
40   /*************************************************************************/
41   /* This function calculates the frequency used. The math is:            */
42   /* T = 1.0 / frequency - Converts to Period                             */
43   /* Ticks = T * 10,000 - Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
44   /* Toggle Rate = Ticks / 2.                                             */
45   /*************************************************************************/
46   unsigned long calculateFrequencyTick(double frequency)
47   {
48       double Period_F = 1.0 / frequency;                            // Period = 1 / Frequency
49       double TicksPerSecond_F = Period_F * 10000.0;                 // Period * 10,000 ticks/second = Ticks.
50       long  TicksPerSecond_L = (long) TicksPerSecond_F;             // Convert to a long from a float
51       long  ToggleRate = TicksPerSecond_L / 2;                      // It takes two toggle to make 1 period
52       return ToggleRate;
53   }
54
55   /*************************************************************************/
56   /* Initialize Task Scheduler                                            */
57   /*************************************************************************/
58   void RunTaskScheduler()
59   {
60       unsigned long int GCD = 1;
61
62       // Declare the local variables. The following are the ticks that the
63       // button and keypads need to run in. Initially they are all zero (disabled)
64       // They are added depending on which part of the lab we are running.
65       unsigned long int Button1Tick = 0;  // Initially disabled
66       unsigned long int Button2Tick = 0;  // Initially disabled
67       unsigned long int Button3Tick = 0;  // Initially disabled
68       unsigned long int KeypadTick  = 0;  // Initially disabled
```

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

```c
/*   jdefo002_lab9_part3.c − 5-13-13
 *   Name & E-mail: Joshua DeForest-Williams jdefo002@ucr.edu
 *   CS Login: jdefo002
 *   Partner(s) Name & E-mail: Ariana Dejaco adeja001@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
```

```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.
```

```
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(tasks[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    }
```

```c
/*    jdefo002_lab9_part4.c - 5-13-13
 *    Name & E-mail: Joshua DeForest-Williams jdefo002@ucr.edu
 *    CS Login: jdefo002
 *    Partner(s) Name & E-mail: Ariana Dejaco Adeja001@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.
    unsigned long int Button1Tick = 0;  // Initially disabled
```

```c
    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.
```

```
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(tasks[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  }
```

```
1   /*    jdefo002_lab9_part5.c - 5-13-13
2    *    Name & E-mail: Joshua DeForest-Williams jdefo002@ucr.edu
3    *    CS Login: jdefo002
4    *    Partner(s) Name & E-mail: Ariana Dejaco Adeja001@ucr.edu
5    *    Lab Section: 22
6    *    Assignment: Lab # 9  Exercise # 5 Challenge
7    *    Exercise Description:Using the ATmega32M-^Rs built in PWM functionality,
8    *    create a piano with seven different tones that can be used to play a
9    *    song.
10   *
11   *    I acknowledge all content contained herein, excluding template or example
12   *    code, is my own original work.
13   */
14
15   #include <avr/io.h>
16   #include <avr/sfr_defs.h>
17   #include <math.h>
18   #include "Bit.h"
19   #include "Buttons.h"
20   #include "KeyPad.h"
21   #include "Speaker_PartChallange.h"
22   #include "Timer0.h"
23   #include "Schedular.h"
24
25   /************************************************************************/
26   /* Initialize the ports                                                 */
27   /************************************************************************/
28   void InitPorts()
29   {
30       // Make PORTD7 as an output. This is connected to the speaker
31       DDRD  = 0x80;
32       PORTD = 0x7F;  // Do NOT use pull up resistor on
33       // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
34       // and PC[7:4] are outputs.
35       DDRC  = 0xF0;
36       PORTC = 0x0F;
37       // Make Port A connected to the buttons. Make them all inputs
38       DDRA = 0x00;
39   }
40
41   /************************************************************************/
42   /* This function calculates the frequency used. The math is:
43   /* T = 1.0 / frequency - Converts to Period                            */
44   /* Ticks = T * 10,000 - Converts to Ticks. Timer0 Ticks 10,000 Ticks/sec*/
45   /* Toggle Rate = Ticks / 2.                                            */
46   /************************************************************************/
47   unsigned long calculateFrequencyTick(double frequency)
48   {
49       double Period_F = 1.0 / frequency;                      // Period = 1 / Frequency
50       double TicksPerSecond_F = Period_F * 10000.0;           // Period * 10,000 ticks/second = Ticks.
51       long   TicksPerSecond_L = (long) TicksPerSecond_F;      // Convert to a long from a float
52       long   ToggleRate = TicksPerSecond_L / 2;               // It takes two toggle to make 1 period
53       return ToggleRate;
54   }
55
56   /************************************************************************/
57   /* Initialize Task Scheduler                                           */
58   /************************************************************************/
59   void RunTaskScheduler()
60   {
61       unsigned long int GCD = 1;
62
63       // Declare the local variables. The following are the ticks that the
64       // button and keypads need to run in. Initially they are all zero (disabled)
65       // They are added depending on which part of the lab we are running.
66       unsigned long int Button1Tick = 0;  // Initially disabled
67       unsigned long int Button2Tick = 0;  // Initially disabled
68       unsigned long int Button3Tick = 0;  // Initially disabled
```

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