

May 13, 13 19:23

Buttons.h

Page 1/3

```

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, Button3State;
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 drive 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)

```

May 13, 13 19:23

Buttons.h

Page 2/3

```

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 drive 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  * This function represents a Tick on the Button3 state machine.
115  *****/
116 int Button3Task(int currentState)
117 {
118     unsigned char button3Value = GetBit(PINA, 2);
119
120     Button3State = (enum Button_States) currentState;
121
122     // State Transitions
123     switch (Button3State)
124     {
125         case ButtonOff:
126             if (button3Value != 0)
127                 Button3State = ButtonOn;
128             else
129                 Button3State = ButtonOff;
130             break;
131
132         case ButtonOn:
133             if (button3Value == 0)
134                 Button3State = ButtonOff;
135             else
136                 Button3State = ButtonOn;
137             break;
138
139         // Placed for completeness. This should NEVER happen.
140         default:
141             Button3State = ButtonOff;
142             break;
143     }

```

May 13, 13 19:23

Buttons.h

Page 3/3

```
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_ */
```

May 13, 13 17:25

KeyPad.h

Page 1/2

```

1  /*
2  * KeyPad.h
3  *
4  * Created: 5/13/2013 6:50:46 AM
5  * Author: Ariana DeJaco
6  */
7
8
9  #ifndef KEYPAD_H_
10 #define KEYPAD_H_
11
12 // The following states are for the buttons
13 enum KeyPad_States {KeyPadOff, Key1, Key2, Key3, Key4, Key5, Key6, Key7, OtherKe
14 y} KeyPadState;
15
16 #define KEY1_FREQUENCY 261.63
17 #define KEY2_FREQUENCY 293.66
18 #define KEY3_FREQUENCY 329.63
19 #define KEY4_FREQUENCY 349.23
20 #define KEY5_FREQUENCY 392.00
21 #define KEY6_FREQUENCY 440.00
22 #define KEY7_FREQUENCY 493.88
23
24 double KeyPadFrequency;
25
26 /*****
27  * Returns '\0' if no key pressed, else returns char '1', '2', ...
28  * If multiple keys pressed, returns leftmost-topmost one
29  * Keypad must be connected to port C
30  * Keypad arrangement
31  *      PC4 PC5 PC6 PC7
32  *      col 1 2 3 4
33  *      row
34  *      PC0 1 1 | 2 | 3 | A
35  *      PC1 2 4 | 5 | 6 | B
36  *      PC2 3 7 | 8 | 9 | C
37  *      PC3 4 * | 0 | # | D
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

```

May 13, 13 17:25

KeyPad.h

Page 2/2

```

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
81     state.
82     // The state is just the keypad value.
83     switch (GetKeypadKey())
84     {
85         case '\0': KeyPadState = KeyPadOff; break;
86         case '1': KeyPadState = Key1; break;
87         case '2': KeyPadState = Key2; break;
88         case '3': KeyPadState = Key3; break;
89         case '4': KeyPadState = Key4; break;
90         case '5': KeyPadState = Key5; break;
91         case '6': KeyPadState = Key6; break;
92         case '7': KeyPadState = Key7; break;
93         default: KeyPadState = OtherKey; break;
94     }
95
96     // Action Code
97     switch (KeyPadState)
98     {
99         case KeyPadOff: KeyPadFrequency = 0.0;
100         break;
101         case Key1: KeyPadFrequency = KEY1_FREQUENCY;
102         break;
103         case Key2: KeyPadFrequency = KEY2_FREQUENCY;
104         break;
105         case Key3: KeyPadFrequency = KEY3_FREQUENCY;
106         break;
107         case Key4: KeyPadFrequency = KEY4_FREQUENCY;
108         break;
109         case Key5: KeyPadFrequency = KEY5_FREQUENCY;
110         break;
111         case Key6: KeyPadFrequency = KEY6_FREQUENCY;
112         break;
113         case Key7: KeyPadFrequency = KEY7_FREQUENCY;
114         break;
115         case OtherKey: KeyPadFrequency = 0.0;
116         break;
117     };
118
119     // Return the current state
120     return KeyPadState;
121 }
122
123 #endif /* KEYPAD_H_ */

```

May 13, 13 17:25

Scheduler.h

Page 1/1

```

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

```

May 13, 13 17:25

Speaker_Part1.h

Page 1/2

```

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;

```

May 13, 13 17:25

Speaker_Part1.h

Page 2/2

```

74 }
75
76 #endif /* SPEAKER_PART1_H_ */

```

May 13, 13 19:51

Speaker_Part2.h

Page 1/2

```

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)) ||

```

May 13, 13 19:51

Speaker_Part2.h

Page 2/2

```

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_ */

```

May 13, 13 17:25

Speaker_Part3.h

Page 1/2

```

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 configure 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

```

May 13, 13 17:25

Speaker_Part3.h

Page 2/2

```

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_ */

```

May 13, 13 17:25

Speaker_Part4.h

Page 1/3

```

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, SpeakerOnWaitForRelease} 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 configure 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 /*****
47  * This function will drive the speaker if the Speaker toggle value
48  * is 0 then D7 is zero otherwise D7 is driven high.
49  *****/
50 void DriveSpeaker()
51 {
52     if ((SpeakerState == SpeakerOn) || (SpeakerState == SpeakerOnWaitForRelease))
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 == ButtonOn))
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

```

May 13, 13 17:25

Speaker_Part4.h

Page 2/3

```

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  * This function represents a Tick on the Button1 state machine.
88  *****/
89 int SpeakerTask(int currentState)
90 {
91     SpeakerState = (enum Speaker_States) currentState;
92
93     // State Transitions
94     switch (SpeakerState)
95     {
96     case SpeakerOff:
97         if (Button1State == ButtonOn)
98         {
99             SpeakerState = SpeakerOnWaitForRelease;
100             DebounceCounter = 0;
101         }
102         break;
103
104     case SpeakerOn:
105         if (Button1State == ButtonOn)
106         {
107             SpeakerState = SpeakerOffWaitForRelease;
108             DebounceCounter = 0;
109         }
110         break;
111
112     case SpeakerOffWaitForRelease:
113         if (DebounceCounter < DEBOUNCE_COUNT_LIMIT)
114             DebounceCounter++; // Wait until we hit DEBOUNCE_COUNT_LIMIT
115
116         else
117         {
118             if (Button1State == ButtonOff)
119             {
120                 SpeakerState = SpeakerOff;
121             }
122             else
123                 DebounceCounter = 0; // Wait some more
124         }
125         break;
126
127     case SpeakerOnWaitForRelease:
128         if (DebounceCounter < DEBOUNCE_COUNT_LIMIT)
129             DebounceCounter++; // Wait until we hit DEBOUNCE_COUNT_LIMIT
130
131         else
132         {
133             if (Button1State == ButtonOff)
134             {
135                 SpeakerState = SpeakerOn;
136             }
137             else
138                 DebounceCounter = 0; // Wait some more
139         }
140     }
141 }

```


May 13, 13 17:25

Speaker_Part4.h

Page 3/3

```
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_ */
```

May 13, 13 17:25

Speaker_PartChallenge.h

Page 1/2

```

1  /*
2  * Speaker_PartChallenge.h
3  *
4  * Created: 5/13/2013 1:55:57 PM
5  * Author: Ariana DeJaco
6  */
7
8  #ifndef SPEAKER_PARTCHALLENGE_H_
9  #define SPEAKER_PARTCHALLENGE_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 configure 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

```

May 13, 13 17:25

Speaker_PartChallenge.h

Page 2/2

```

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

```

May 13, 13 17:25

Timer0.h

Page 1/1

```

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 TimerISR()
56 ISR(TIMER0_COMP_vect)
57 {
58     // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn settings)
59     _avr_timer_cntcurr--; // Count down to 0 rather than up
60     if (_avr_timer_cntcurr == 0) { // results in a more efficient compare
61         TimerISR(); // Call the ISR that the
62         user uses
63         _avr_timer_cntcurr = _avr_timer_M;
64     }
65 }
66 #endif //TIMER_H

```

May 13, 13 20:49

adeja001_lab9_part1.c

Page 1/3

```

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 "Scheduler.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; // Conve
52     rt to a long from a float
53     long ToggleRate = TicksPerSecond_L / 2;
54     // 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
67     led)
68
69     // They are added depending on which part of the lab we are running.
70     unsigned long int Button1Tick = 0; // Initially disabled
71     unsigned long int Button2Tick = 0; // Initially disabled
72     unsigned long int Button3Tick = 0; // Initially disabled
73     unsigned long int KeypadTick = 0; // Initially disabled

```

May 13, 13 20:49

adeja001_lab9_part1.c

Page 2/3

```

68 // Determine the real ticks that we need to run the button and keypad -
69 // Based on the lab part that we are running.
70 // Determine the real ticks that we need to run the button and keypad -
71 // Based on the lab part that we are running.
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_PARTCHALLENGE_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
94 n at the fastest rate (Twice the fastest button).
95 unsigned long int Button1Period = Button1Tick/GCD;
96 unsigned long int Button2Period = Button2Tick/GCD;
97 unsigned long int Button3Period = Button3Tick/GCD;
98 unsigned long int KeypadPeriod = KeypadTick/GCD;
99
100 // Declare an array of tasks
101 static task task1, task2, task3, task4, task5;
102 task *tasks[] = { &task1, &task2, &task3, &task4, &task5};
103 const unsigned short numTasks = sizeof(tasks)/sizeof(task*);
104
105 // Task 1
106 task1.state = (int) SpeakerOff; //Task initial state.
107 task1.period = SpeakerPeriod; //Task Period.
108 task1.elapsedTime = SpeakerPeriod; //Task current elapsed time.
109 task1.TickFct = &SpeakerTask; //Function pointer for the tick.
110
111 // Task 2
112 task2.state = (int) ButtonOff; //Task initial state.
113 task2.period = Button1Period; //Task Period.
114 task2.elapsedTime = Button1Period; //Task current elapsed time.
115 task2.TickFct = &Button1Task; //Function pointer for the tick.
116
117 // Task 3
118 task3.state = (int) ButtonOff; //Task initial state.
119 task3.period = Button2Period; //Task Period.
120 task3.elapsedTime = Button2Period; //Task current elapsed time.
121 task3.TickFct = &Button2Task; //Function pointer for the tick.
122
123 // Task 4
124 task4.state = (int) ButtonOff; //Task initial state.
125 task4.period = Button3Period; //Task Period.
126 task4.elapsedTime = Button3Period; //Task current elapsed time.
127 task4.TickFct = &Button3Task; //Function pointer for the tick.
128
129 // Task 5
130 task5.state = (int) KeypadOff; //Task initial state.
131 task5.period = KeypadPeriod; //Task Period.
132 task5.elapsedTime = KeypadPeriod; //Task current elapsed time.
133 task5.TickFct = &KeypadTask; //Function pointer for the tick.
134
135 // Set the timer and turn it on
136 TimerSet(GCD);
137 TimerOn();

```

May 13, 13 20:49

adeja001_lab9_part1.c

Page 3/3

```

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
153                 // Reset the elapsed time for next tick.
154                 tasks[i]->elapsedTime = 0;
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 }

```

May 13, 13 20:48

adeja001_lab9_part2.c

Page 1/3

```

1  /*      adeja001_lab9_part2.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 # 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 "SpeakerPart2.h"
21  #include "Timer0.h"
22  #include "Scheduler.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  /*
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;
50      // Period = 1 / Frequency
51      double TicksPerSecond_F = Period_F * 10000.0; // Perio
52      d * 10,000 ticks/second = Ticks.
53      long TicksPerSecond_L = (long) TicksPerSecond_F; // Conve
54      rt to a long from a float
55      long ToggleRate = TicksPerSecond_L / 2;
56      // It takes two toggle to make 1 period
57      return ToggleRate;
58  }
59
60  /******
61  /* Initialize Task Scheduler
62  /******
63  void RunTaskScheduler()
64  {
65      unsigned long int GCD = 1;
66
67      // Declare the local variables. The following are the ticks that the
68      // button and keypads need to run in. Initially they are all zero (disab
69      led)
70
71      // They are added depending on which part of the lab we are running.
72      unsigned long int Button1Tick = 0; // Initially disabled
73      unsigned long int Button2Tick = 0; // Initially disabled
74      unsigned long int Button3Tick = 0; // Initially disabled

```

May 13, 13 20:48

adeja001_lab9_part2.c

Page 2/3

```

68      unsigned long int KeypadTick = 0; // Initially disabled
69
70      //*****
71      // Determine the real ticks that we need to run the button and keypad -
72      // If any..
73      // Based on the lab part that we are running.
74      //*****
75      #ifndef SPEAKER_PART1_H_
76          Button1Tick = calculateFrequencyTick(Button1Frequency);
77      #endif
78      #ifndef SPEAKER_PART2_H_
79          Button1Tick = calculateFrequencyTick(Button1Frequency);
80          Button2Tick = calculateFrequencyTick(Button2Frequency);
81          Button3Tick = calculateFrequencyTick(Button3Frequency);
82      #endif
83      #ifndef SPEAKER_PART3_H_
84          Button1Tick = 100; // Good number for button sampling....
85      #endif
86      #ifndef 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      #ifndef SPEAKER_PARTCHALLENGE_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 drive
97      n at the fastest rate (Twice the fastest button).
98      unsigned long int Button1Period = Button1Tick/GCD;
99      unsigned long int Button2Period = Button2Tick/GCD;
100      unsigned long int Button3Period = Button3Tick/GCD;
101      unsigned long int KeyPadPeriod = KeypadTick/GCD;
102
103      // Declare an array of tasks
104      static task_t task1, task2, task3, task4, task5;
105      task_t *tasks[] = { &task1, &task2, &task3, &task4, &task5 };
106      const unsigned short numTasks = sizeof(tasks)/sizeof(task_t*);
107
108      // Task 1
109      task1.state = (int) SpeakerOff; //Task initial state.
110      task1.period = SpeakerPeriod; //Task Period.
111      task1.elapsedTime = SpeakerPeriod; //Task current elapsed time.
112      task1.TickFct = &SpeakerTask; //Function pointer for the tick.
113
114      // Task 2
115      task2.state = (int) ButtonOff; //Task initial state.
116      task2.period = Button1Period; //Task Period.
117      task2.elapsedTime = Button1Period; //Task current elapsed time.
118      task2.TickFct = &Button1Task; //Function pointer for the tick.
119
120      // Task 3
121      task3.state = (int) ButtonOff; //Task initial state.
122      task3.period = Button2Period; //Task Period.
123      task3.elapsedTime = Button2Period; //Task current elapsed time.
124      task3.TickFct = &Button2Task; //Function pointer for the tick.
125
126      // Task 4
127      task4.state = (int) ButtonOff; //Task initial state.
128      task4.period = Button3Period; //Task Period.
129      task4.elapsedTime = Button3Period; //Task current elapsed time.
130      task4.TickFct = &Button3Task; //Function pointer for the tick.
131
132      // Task 5
133      task5.state = (int) KeyPadOff; //Task initial state.
134      task5.period = KeyPadPeriod; //Task Period.
135      task5.elapsedTime = KeyPadPeriod; //Task current elapsed time.
136      task5.TickFct = &KeyPadTask; //Function pointer for the tick.
137
138      // Set the timer and turn it on

```

May 13, 13 20:48

adeja001_lab9_part2.c

Page 3/3

```

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
155                     // Reset the elapsed time for next tick.
156                     tasks[i]->elapsedTime = 0;
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     }

```

May 13, 13 20:47

adeja001_lab9_part3.c

Page 1/3

```

1  /*      adeja001_lab9_part3.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 # 3
7  *      Exercise Description: This section will be using the PWM functionality
8  *      on timer/counter 2 of the ATmega32. Since we will be using a different
9  *      timer than the one used in previous labs, the changes made to TCCR0
10 *      and OCR0 can be reverted back to their original values.
11 *
12 *      I acknowledge all content contained herein, excluding template or example
13 *      code, is my own original work.
14 */
15
16 #include <avr/io.h>
17 #include <avr/sfr_defs.h>
18 #include <math.h>
19 #include "Bit.h"
20 #include "Buttons.h"
21 #include "KeyPad.h"
22 #include "Speaker_Part3.h"
23 #include "Timer0.h"
24 #include "Scheduler.h"
25
26 /*****
27 /* Initialize the ports
28 *****/
29 void InitPorts()
30 {
31     // Make PORTD7 as an output. This is connected to the speaker
32     DDRD = 0x80;
33     PORTD = 0x7F; // Do NOT use pull up resistor on
34     // Make PORTC Connected to the KeyPad. PC[3:0] are inputs
35     // and PC[7:4] are outputs.
36     DDRC = 0xF0;
37     PORTC = 0x0F;
38     // Make Port A connected to the buttons. Make them all inputs
39     DDRA = 0x00;
40 }
41
42 /*****
43 /* This function calculates the frequency used. The math is:
44 */
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;
52     // Period = 1 / Frequency
53     double TicksPerSecond_F = Period_F * 10000.0; // Perio
54     d * 10,000 ticks/second = Ticks.
55     long TicksPerSecond_L = (long) TicksPerSecond_F; // Conve
56     rt to a long from a float
57     long ToggleRate = TicksPerSecond_L / 2;
58     // It takes two toggle to make 1 period
59     return ToggleRate;
60 }
61
62 /*****
63 /* Initialize Task Scheduler
64 *****/
65 void RunTaskScheduler()
66 {
67     unsigned long int GCD = 1;
68
69     // Declare the local variables. The following are the ticks that the
70     // button and keypads need to run in. Initially they are all zero (disab
71     led)
72     // They are added depending on which part of the lab we are running.
73     unsigned long int Button1Tick = 0; // Initially disabled

```

May 13, 13 20:47

adeja001_lab9_part3.c

Page 2/3

```

68     unsigned long int Button2Tick = 0; // Initially disabled
69     unsigned long int Button3Tick = 0; // Initially disabled
70     unsigned long int KeypadTick = 0; // Initially disabled
71
72     // Determine the real ticks that we need to run the button and keypad -
73     If any..
74     // Based on the lab part that we are running.
75     // Determine the real ticks that we need to run the button and keypad -
76     // Based on the lab part that we are running.
77     // Determine the real ticks that we need to run the button and keypad -
78     // Based on the lab part that we are running.
79     // Determine the real ticks that we need to run the button and keypad -
80     // Based on the lab part that we are running.
81     // Determine the real ticks that we need to run the button and keypad -
82     // Based on the lab part that we are running.
83     // Determine the real ticks that we need to run the button and keypad -
84     // Based on the lab part that we are running.
85     // Determine the real ticks that we need to run the button and keypad -
86     // Based on the lab part that we are running.
87     // Determine the real ticks that we need to run the button and keypad -
88     // Based on the lab part that we are running.
89     // Determine the real ticks that we need to run the button and keypad -
90     // Based on the lab part that we are running.
91     // Determine the real ticks that we need to run the button and keypad -
92     // Based on the lab part that we are running.
93     // Determine the real ticks that we need to run the button and keypad -
94     // Based on the lab part that we are running.
95     // Determine the real ticks that we need to run the button and keypad -
96     // Based on the lab part that we are running.
97     // Determine the real ticks that we need to run the button and keypad -
98     // Based on the lab part that we are running.
99     // Determine the real ticks that we need to run the button and keypad -
100    // Based on the lab part that we are running.
101    // Determine the real ticks that we need to run the button and keypad -
102    // Based on the lab part that we are running.
103    // Determine the real ticks that we need to run the button and keypad -
104    // Based on the lab part that we are running.
105    // Determine the real ticks that we need to run the button and keypad -
106    // Based on the lab part that we are running.
107    // Determine the real ticks that we need to run the button and keypad -
108    // Based on the lab part that we are running.
109    // Determine the real ticks that we need to run the button and keypad -
110    // Based on the lab part that we are running.
111    // Determine the real ticks that we need to run the button and keypad -
112    // Based on the lab part that we are running.
113    // Determine the real ticks that we need to run the button and keypad -
114    // Based on the lab part that we are running.
115    // Determine the real ticks that we need to run the button and keypad -
116    // Based on the lab part that we are running.
117    // Determine the real ticks that we need to run the button and keypad -
118    // Based on the lab part that we are running.
119    // Determine the real ticks that we need to run the button and keypad -
120    // Based on the lab part that we are running.
121    // Determine the real ticks that we need to run the button and keypad -
122    // Based on the lab part that we are running.
123    // Determine the real ticks that we need to run the button and keypad -
124    // Based on the lab part that we are running.
125    // Determine the real ticks that we need to run the button and keypad -
126    // Based on the lab part that we are running.
127    // Determine the real ticks that we need to run the button and keypad -
128    // Based on the lab part that we are running.
129    // Determine the real ticks that we need to run the button and keypad -
130    // Based on the lab part that we are running.
131    // Determine the real ticks that we need to run the button and keypad -
132    // Based on the lab part that we are running.
133    // Determine the real ticks that we need to run the button and keypad -
134    // Based on the lab part that we are running.
135    // Determine the real ticks that we need to run the button and keypad -
136    // Based on the lab part that we are running.

```


May 13, 13 20:47

adeja001_lab9_part3.c

Page 3/3

```

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
157                 // Reset the elapsed time for next tick.
158                 tasks[i]->elapsedTime = 0;
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 }

```

May 13, 13 20:44

adeja001_lab9_part4.c

Page 1/3

```

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 "Scheduler.h"
26
27 /***** Initialize the ports *****/
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 /***** 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 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; // Conve
52     rt to a long from a float
53     long ToggleRate = TicksPerSecond_L / 2;
54     // It takes two toggle to make 1 period
55     return ToggleRate;
56 }
57
58 /***** Initialize Task Scheduler *****/
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 (disab
65     led)
66     // They are added depending on which part of the lab we are running.
67

```

May 13, 13 20:44

adeja001_lab9_part4.c

Page 2/3

```

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     // Determine the real ticks that we need to run the button and keypad -
74     // 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_PARTCHALLENGE_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
99     n at the fastest rate (Twice the fastest button).
100     unsigned long int Button1Period = Button1Tick/GCD;
101     unsigned long int Button2Period = Button2Tick/GCD;
102     unsigned long int Button3Period = Button3Tick/GCD;
103     unsigned long int KeypadPeriod = KeypadTick/GCD;
104
105     // Declare an array of tasks
106     static task task1, task2, task3, task4, task5;
107     task *tasks[] = { &task1, &task2, &task3, &task4, &task5 };
108     const unsigned short numTasks = sizeof(tasks)/sizeof(task*);
109
110     // Task 1
111     task1.state = (int) SpeakerOff; //Task initial state.
112     task1.period = SpeakerPeriod; //Task Period.
113     task1.elapsedTime = SpeakerPeriod; //Task current elapsed time.
114     task1.TickFct = &SpeakerTask; //Function pointer for the tick.
115
116     // Task 2
117     task2.state = (int) ButtonOff; //Task initial state.
118     task2.period = Button1Period; //Task Period.
119     task2.elapsedTime = Button1Period; //Task current elapsed time.
120     task2.TickFct = &Button1Task; //Function pointer for the tick.
121
122     // Task 3
123     task3.state = (int) ButtonOff; //Task initial state.
124     task3.period = Button2Period; //Task Period.
125     task3.elapsedTime = Button2Period; //Task current elapsed time.
126     task3.TickFct = &Button2Task; //Function pointer for the tick.
127
128     // Task 4
129     task4.state = (int) ButtonOff; //Task initial state.
130     task4.period = Button3Period; //Task Period.
131     task4.elapsedTime = Button3Period; //Task current elapsed time.
132     task4.TickFct = &Button3Task; //Function pointer for the tick.
133
134     // Task 5
135     task5.state = (int) KeypadOff; //Task initial state.
136     task5.period = KeypadPeriod; //Task Period.
137     task5.elapsedTime = KeypadPeriod; //Task current elapsed time.

```

May 13, 13 20:44

adeja001_lab9_part4.c

Page 3/3

```

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
158                     // Reset the elapsed time for next tick.
159                     tasks[i]->elapsedTime = 0;
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     }

```

May 13, 13 20:43

adeja001_lab9_part5.c

Page 1/3

```

1  /*      adeja001_lab9_part5.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 # 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_PartChallenge.h"
22 #include "Timer0.h"
23 #include "Scheduler.h"
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 /***** This function calculates the frequency used. The math is:
41 */
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;
49     // Period = 1 / Frequency
50     double TicksPerSecond_F = Period_F * 10000.0; // Perio
51     d * 10,000 ticks/second = Ticks. // Conve
52     long TicksPerSecond_L = (long) TicksPerSecond_F;
53     rt to a long from a float
54     long ToggleRate = TicksPerSecond_L / 2;
55     // It takes two toggle to make 1 period
56     return ToggleRate;
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
67     led)
68
69     // They are added depending on which part of the lab we are running.
70     unsigned long int Button1Tick = 0; // Initially disabled
71     unsigned long int Button2Tick = 0; // Initially disabled

```

May 13, 13 20:43

adeja001_lab9_part5.c

Page 2/3

```

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

```

May 13, 13 20:43

adeja001_lab9_part5.c

Page 3/3

```

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          // Process_LCD_Task(1);
162          while(!TimerFlag);
163          TimerFlag = 0;
164      }
165  }
166  }
167
168  int main(void)
169  {
170      InitPorts();
171      InitSpeaker();
172      RunTaskScheduler();
173  }

```