

Apr 29, 13 20:57

jdefo002_lab6_part1.c

Page 1/3

```

1  /* jdefo002_lab6_part1.c - April 29, 2013
2  * Name: Joshua Deforest-Williams E-mail jdefo002@ucr.edu
3  * CS Login: jdefo002
4  * Partner Name: Ariana DeJaco Email adeja001@ucr.edu
5  * Lab Section: 022
6  * Assignment: Lab#6 Exercise#1
7  * Exercise Description: Create a synchSM to blink three LEDs
8  * connected to PB0, PB1, and PB2 in sequence, 1 second each.
9  * Implement that synchSM in C using the method defined in class.
10 * In addition to demoing your program, you will need to show that
11 * your code adheres entirely to the method with no variations.
12 */
13
14 #include <avr/io.h>
15 #include <avr/interrupt.h>
16
17 // Current Port Definitions
18 #define LED_DDR          DDRB
19 #define LED_INPORT       PINB
20 #define LED_OUTPORT      PORTB
21 #define SENSOR_DDR       DDRA
22 #define SENSOR_INPORT    PINA
23 #define SENSOR_OUTPORT   PORTA
24 #define UNSEDC_DDR       DDRC
25 #define UNSEDC_PIN       PINC
26 #define UNSEDC_PORT      PORTC
27 #define UNSEDD_DDR       DDRD
28 #define UNSEDD_PIN       PIND
29 #define UNSEDD_PORT      PORTD
30
31 // Additional macros not defines in sfr_defs.h
32 #define SET_PORT_BIT(OUTPORT, BIT)      OUTPORT |= (1 << BIT)
33 #define CLEAR_PORT_BIT(OUTPORT, BIT)    OUTPORT &= ~(1 << BIT)
34
35 //DDRA: Configures each of port A's physical pins to input (0) or output(1)
36 //PORTA: Writing to this register writes the port's physical pins
37 // (Write only)
38 //PINA: Reading this register reads the values of the port's physical pins
39 // (Read only)
40
41 volatile unsigned char TimerFlag = 0; // TimerISR() sets this to 1. C programmer
42   should clear to 0.
43
44 // Internal variables for mapping AVR's ISR to our cleaner TimerISR model.
45 unsigned long _avr_timer_M = 1; // Start count from here, down to 0. Default 1 m
46 s.
47 unsigned long _avr_timer_cntcurr = 0; // Current internal count of lms ticks
48
49 void TimerOn()
50 {
51     // AVR timer/counter controller register TCCR0
52     TCCR0 = 0x0B; // bit3bit6=10: CTC mode (clear timer on compare)
53     // bit2bit1bit0=011: pre-scaler /64
54     // 00001011: 0x0B
55     // SO, 8 MHz clock or 8,000,000 /64 = 125,000 ticks/s
56     // Thus, TCNT0 register will count at 125,000 ticks/s
57
58     // AVR output compare register OCR0.
59     OCR0 = 125; // Timer interrupt will be generated when TCNT0==OCR0
60     // We want a 1 ms tick. 0.001 s * 125,000 ticks/s = 125
61     // So when TCNT0 register equals 125,
62     // 1 ms has passed. Thus, we compare to 125.
63     // AVR timer interrupt mask register
64     TIMSK = 0x02; // bit1: OCIE0 -- enables compare match interrupt
65
66     //Initialize avr counter
67     TCNT0=0;
68
69     _avr_timer_cntcurr = _avr_timer_M;
70     // TimerISR will be called every _avr_timer_cntcurr milliseconds
71
72     //Enable global interrupts
73     SREG |= 0x80; // 0x80: 1000000

```

Apr 29, 13 20:57

jdefo002_lab6_part1.c

Page 2/3

```

72 }
73
74 void TimerOff()
75 {
76     TCCR0 = 0x00; // bit2bit1bit0=000: timer off
77 }
78
79 void TimerISR()
80 {
81     TimerFlag = 1;
82 }
83
84 // In our approach, the C programmer does not touch this ISR, but rather TimerIS
85 R()
86 ISR(TIMER0_COMP_vect)
87 {
88     // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn se
89 ttings)
90     _avr_timer_cntcurr--; // Count down to 0 rather than up to TOP
91     if (_avr_timer_cntcurr == 0)
92     { // results in a more efficient compare
93         TimerISR(); // Call the ISR that the user uses
94         _avr_timer_cntcurr = _avr_timer_M;
95     }
96 }
97
98 // Set TimerISR() to tick every M ms
99 void TimerSet(unsigned long M)
100 {
101     _avr_timer_M = M;
102     _avr_timer_cntcurr = _avr_timer_M;
103 }
104
105 enum BL_States { Initial, BL_LedOff, BL_LedOneOn, BL_LedTwoOn, BL_LedThreeOn } B
106 L_State;
107
108 void TickFct_Blink()
109 {
110     switch ( BL_State )
111     { //Transitions
112         case Initial:
113             BL_State = BL_LedOff; //Initial state
114             break;
115         case BL_LedOff:
116             BL_State = BL_LedOneOn;
117             break;
118         case BL_LedOneOn:
119             BL_State = BL_LedTwoOn;
120             break;
121         case BL_LedTwoOn:
122             BL_State = BL_LedThreeOn;
123             break;
124         case BL_LedThreeOn:
125             BL_State = BL_LedOff;
126             break;
127         default:
128             BL_State = -1;
129             break;
130     }
131
132     switch (BL_State)
133     { //State actions
134         case BL_LedOff:
135             LED_OUTPORT = 0x00;
136             break;
137         case BL_LedOneOn:
138             LED_OUTPORT = 0x01;
139             break;
140         case BL_LedTwoOn:
141             LED_OUTPORT = 0x02;
142             break;
143         case BL_LedThreeOn:
144             LED_OUTPORT = 0x04;
145             break;

```

Apr 29, 13 20:57

jdefo002_lab6_part1.c

Page 3/3

```
142         break;
143     default:
144         break;
145     }
146 }
147
148 void main()
149 {
150     LED_DDR = 0xFF; // Set port B to output
151     LED_OUTPORT = 0x00; // Init port B to 0s
152
153     TimerSet(1000);
154     TimerOn();
155
156     BL_State = Initial; // Indicates initial tick function call
157
158     while(1)
159     {
160         TickFct_Blink(); // Execute one tick of the BL synchSM
161         while (!TimerFlag){} // Wait for BLM-^Rs period
162         TimerFlag = 0;
163     }
164 }
```

Apr 29, 13 20:58

jdefo002_lab6_part2.c

Page 1/5

```

1  /* jdefo002_lab6_part2.c - April 29, 2013
2  * Name: Joshua DeForest-Williams E-mail jdefo002@ucr.edu
3  * CS Login: jdefo002
4  * Partner Name: Ariana DeJaco Email adeja001@ucr.edu
5  * Lab Section: 022
6  * Assignment: Lab#6 Exercise#2
7  * Exercise Description: (From an earlier lab) Buttons are connected
8  * to PA0 and PA1. Output PORTC drives the 7-segment display initially
9  * displaying 0. Pressing PA0 increments the display (stopping at 9).
10 * Pressing PA1 decrements the display (stopping at 0). If both buttons
11 * are depressed (even if not initially simultaneously), the display
12 * resets to 0. If a button is held, then the display continues to
13 * increment (or decrement) at a rate of once per second. Use a
14 * synchronous state machine captured in C.
15 */
16
17 #define F_CPU 8000000
18 // Interrupt.h is needed otherwise it will NOT work!!!!
19
20 #include <avr/io.h>
21 #include <avr/interrupt.h>
22 #include <util/delay.h>
23 #include <avr/sfr_defs.h>
24
25 // Bit-access function
26 unsigned char GetBit(unsigned char x, unsigned char k)
27 {
28     return ((x & (0x01 << k)) != 0);
29 }
30
31 // Current Port Definitions
32 #define LED_DDR          DDRC
33 #define LED_INPORT       PINC
34 #define LED_OUTPORT      PORTC
35 #define SENSOR_DDR      DDRA
36 #define SENSOR_INPORT    PINA
37 #define SENSOR_OUTPORT   PORTA
38 #define UNUSEDB_DDR      DDRB
39 #define UNUSEDB_PIN      PINB
40 #define UNUSEDB_PORT     PORTB
41 #define UNUSEDDB_DDR      DDRD
42 #define UNUSEDDB_PIN      PIND
43 #define UNUSEDDB_PORT     PORTD
44
45 // Additional macros not defines in sfr_defs.h
46 #define SET_PORT_BIT(OUTPORT, BIT)    OUTPORT |= (1 << BIT)
47 #define CLEAR_PORT_BIT(OUTPORT, BIT)  OUTPORT &= ~(1 << BIT)
48
49 //DDRA: Configures each of port A's physical pins to input (0) or output(1)
50 //PORTA: Writing to this register writes the port's physical pins
51 // (Write only)
52 //PINA: Reading this register reads the values of the port's physical pins
53 // (Read only)
54
55 volatile unsigned char TimerFlag = 0; // TimerISR() sets this to 1. C programmer
56     should clear to 0.
57
58 // Internal variables for mapping AVR's ISR to our cleaner TimerISR model.
59 volatile unsigned long _avr_timer_M = 1; // Start count from here, down to 0. De
60     fault 1 ms.
61 volatile unsigned long _avr_timer_cntcurr = 0; // Current internal count of 1ms
62     ticks
63
64 void TimerOn()
65 {
66     // AVR timer/counter controller register TCCR0
67     TCCR0 = 0x0B; // bit3bit6=10: CTC mode (clear timer on compare)
68     // bit2bit1bit0=011: pre-scaler /64
69     // 00001011: 0x0B
70     // SO, 8 MHz clock or 8,000,000 /64 = 125,000 ticks/s
71     // Thus, TCNT0 register will count at 125,000 ticks/s
72
73     // AVR output compare register OCR0.

```

Apr 29, 13 20:58

jdefo002_lab6_part2.c

Page 2/5

```

71     OCR0 = 125; // Timer interrupt will be generated when TCNT0==OCR0
72     // We want a 1 ms tick. 0.001 s * 125,000 ticks/s = 125
73     // So when TCNT0 register equals 125,
74     // 1 ms has passed. Thus, we compare to 125.
75     // AVR timer interrupt mask register
76     TIMSK = 0x02; // bit1: OCIE0 -- enables compare match interrupt
77
78     //Initialize avr counter
79     TCNT0=0;
80
81     _avr_timer_cntcurr = _avr_timer_M;
82     // TimerISR will be called every _avr_timer_cntcurr milliseconds
83
84     //Enable global interrupts
85     SREG |= 0x80; // 0x80: 10000000
86 }
87
88 void TimerOff()
89 {
90     TCCR0 = 0x00; // bit2bit1bit0=000: timer off
91 }
92
93 void TimerISR()
94 {
95     TimerFlag = 1;
96 }
97
98 // In our approach, the C programmer does not touch this ISR, but rather TimerIS
99 R()
100 ISR(TIMERO_COMP_vect)
101 {
102     // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn se
103     ttings)
104     _avr_timer_cntcurr--; // Count down to 0 rather than up to TOP
105     if (_avr_timer_cntcurr == 0)
106     { // results in a more efficient compare
107         TimerISR(); // Call the ISR that the user uses
108         _avr_timer_cntcurr = _avr_timer_M;
109     }
110 }
111
112 // Set TimerISR() to tick every M ms
113 void TimerSet(unsigned long M)
114 {
115     _avr_timer_M = M;
116     _avr_timer_cntcurr = _avr_timer_M;
117 }
118
119 enum Counter_States { InitReset, WaitForButtonPress, Increment, Decrement, Error
120     State } CounterState;
121
122 unsigned char TckFct_Counter(unsigned char inputData, unsigned char LedValue)
123 {
124     // Variable we are returning
125     unsigned char tempLedValue = LedValue;
126     unsigned char checkIncrement = 0;
127     unsigned char checkDecrement = 0;
128     unsigned char i = 0;
129
130     switch(CounterState)
131     {
132         //Transitions
133         case InitReset: // Initial Transition
134             CounterState = WaitForButtonPress;
135             break;
136         case WaitForButtonPress: // Initial Transition
137             if(inputData == 0x00)
138             {
139                 CounterState = WaitForButtonPress;
140             }
141             else if(inputData == 0x01)
142             {
143                 CounterState = Increment;
144             }

```

Apr 29, 13 20:58

jdefo002_lab6_part2.c

Page 3/5

```

141     }
142     else if(inputData == 0x02)
143     {
144         CounterState = Decrement;
145     }
146     else if(inputData == 0x03)
147     {
148         CounterState = InitReset;
149     }
150     else
151     {
152         CounterState = ErrorState;
153     }
154     break;
155 case Increment:
156     CounterState = WaitForButtonPress;
157     break;
158 case Decrement:
159     CounterState = WaitForButtonPress;
160     break;
161 case ErrorState:
162     break;
163 default:
164     CounterState = ErrorState;
165     break;
166 }
167
168 switch(CounterState)
169 { // Actions
170     case Increment:
171         if(tempLedValue < 9)
172         {
173             tempLedValue++;
174         }
175
176         checkIncrement = GetBit(SENSOR_INPORT, 0);
177         if(checkIncrement == 0)
178         {
179             for(i = 0; i < 5; i++)
180             {
181                 while(!TimerFlag){}
182                 TimerFlag = 0;
183             }
184             break;
185         }
186     case Decrement:
187         if(tempLedValue > 0)
188         {
189             tempLedValue--;
190         }
191
192         checkDecrement = GetBit(SENSOR_INPORT, 1);
193         if(checkDecrement == 0)
194         {
195             for(i = 0; i < 5; i++)
196             {
197                 while(!TimerFlag){}
198                 TimerFlag = 0;
199             }
200             break;
201         }
202     case InitReset:
203         tempLedValue = 0x00;
204         break;
205     case WaitForButtonPress:
206         break;
207     case ErrorState:
208         tempLedValue = 0xFF;
209         break;
210     default:
211         break;
212 }
213 return tempLedValue;

```

Apr 29, 13 20:58

jdefo002_lab6_part2.c

Page 4/5

```

214 }
215
216 unsigned char Write7Seg(unsigned char x)
217 {
218     unsigned char segvalue = 0;
219
220     switch(x)
221     {
222         case 0:
223             segvalue = 0x3F;
224             break;
225         case 1:
226             segvalue = 0x0C;
227             break;
228         case 2:
229             segvalue = 0x5B;
230             break;
231         case 3:
232             segvalue = 0x5E;
233             break;
234         case 4:
235             segvalue = 0x6C;
236             break;
237         case 5:
238             segvalue = 0x76;
239             break;
240         case 6:
241             segvalue = 0x77;
242             break;
243         case 7:
244             segvalue = 0x1C;
245             break;
246         case 8:
247             segvalue = 0x7F;
248             break;
249         case 9:
250             segvalue = 0x7C;
251             break;
252         default:
253             segvalue = 0x73;
254             break;
255     }
256
257     return segvalue;
258 }
259
260 int main(void)
261 {
262     SENSOR_DDR = 0x00;
263     LED_DDR = 0xFF;
264     UNSEDB_DDR = 0x00;
265     UNSEDD_DDR = 0x00;
266
267     // Initialize LEDs to off
268     LED_OUTPORT = 0x00;
269     unsigned char ButtonValue = 0;
270     unsigned char CurrentLEDValue = 0;
271     unsigned char SevenSegValue = 0;
272     unsigned char PreviousButtonValue = 0;
273
274     // The Timer is set to 500 ms because in the increment/decrement state i
275     t has // to go back to the wait for button press state. Therefore, there are t
276     wo // state transitions, each take up 500ms making a total of 1 second.
277     TimerSet(100);
278     TimerOn();
279
280     while(1)
281     {
282         // Wait for 1 second before advancing the state machine. While w
283         aiting see if // any buttons have been pressed. Save the last button pressed v

```

Apr 29, 13 20:58

jdefo002_lab6_part2.c

Page 5/5

```

284         ButtonValue = 0;    // Assume no buttons have been pressed
285         TimerFlag = 0;      // Reset the timer flag. It will be set by t
he timer0 interrupt.
286
287         // Wait until the timer0 interrupt executes (Once every second)
288         while (!TimerFlag)
289         {
290             // If some buttons have been pressed then save what has
been pressed.
291             if ((~SENSOR_INPORT & 0x03) != 0)
292             {
293                 ButtonValue = (~SENSOR_INPORT & 0x03);
294             }
295         }
296         // Now that 1 second is up we can advance the state machine
297         CurrentLEDValue = TckFct_Counter(ButtonValue, CurrentLEDValue);
298         // Write the reset to the seven segment display
299         SevenSegValue = Write7Seg(CurrentLEDValue);
300         LED_OUTPORT = SevenSegValue;
301     }
302 }

```

Apr 29, 13 20:59

jdefo002_lab6_part3.c

Page 1/4

```

1  /* jdefo002_lab6_part3.c - April 29, 2013
2  * Name: Joshua DeForest-Williams E-mail jdefo002@ucr.edu
3  * CS Login: jdefo002
4  * Partner Name: Ariana DeJaco Email adeja001@ucr.edu
5  * Lab Section: 022
6  * Assignment: Lab#6 Exercise#3/challenge
7  * Exercise Description: (Challenge) Create a simple game that
8  * requires pressing a button on PA0 while the middle of three
9  * LEDs on PB0, PB1, and PB2 is lit. The LEDs light for 300 ms
10 * each in sequence. When the button is pressed, the currently
11 * lit LED stays lit. Pressing the button again restarts the game.
12 */
13
14 // Interrupt.h is needed otherwise it will NOT work!!!!
15
16 #include <avr/io.h>
17 #include <avr/interrupt.h>
18 #include <util/delay.h>
19 #include <avr/sfr_defs.h>
20
21 // Bit-access function
22 unsigned char GetBit(unsigned char x, unsigned char k)
23 {
24     return ((x & (0x01 << k)) != 0);
25 }
26
27 // Current Port Definitions
28 #define LED_DDR          DDRB
29 #define LED_INPORT       PINB
30 #define LED_OUTPORT      PORTB
31 #define SENSOR_DDR       DDRA
32 #define SENSOR_INPORT    PINA
33 #define SENSOR_OUTPORT   PORTA
34 #define UNUSED_C_DDR     DDRC
35 #define UNUSED_C_PIN     PINC
36 #define UNUSED_C_PORT    PORTC
37 #define UNUSED_D_DDR     DDRD
38 #define UNUSED_D_PIN     PIND
39 #define UNUSED_D_PORT    PORTD
40
41 // Additional macros not defines in sfr_defs.h
42 #define SET_PORT_BIT(PORT, BIT)      PORT |= (1 << BIT)
43 #define CLEAR_PORT_BIT(PORT, BIT)    PORT &= ~(1 << BIT)
44
45 //DDRA: Configures each of port A's physical pins to input (0) or output(1)
46 //PORTA: Writing to this register writes the port's physical pins
47 // (Write only)
48 //PINA: Reading this register reads the values of the port's physical pins
49 // (Read only)
50
51 volatile unsigned char TimerFlag = 0; // TimerISR() sets this to 1. C programmer
52   should clear to 0.
53
54 // Internal variables for mapping AVR's ISR to our cleaner TimerISR model.
55 volatile unsigned long _avr_timer_M = 1; // Start count from here, down to 0. De
56   fault 1 ms.
57 volatile unsigned long _avr_timer_cntcurr = 0; // Current internal count of 1ms
58   ticks
59
60 void TimerOn()
61 {
62     // AVR timer/counter controller register TCCR0
63     TCCR0 = 0x0B; // bit3bit6=10: CTC mode (clear timer on compare)
64     // bit2bit1bit0=011: pre-scaler /64
65     // 00001011: 0x0B
66     // SO, 8 MHz clock or 8,000,000 /64 = 125,000 ticks/s
67     // Thus, TCNT0 register will count at 125,000 ticks/s
68
69     // AVR output compare register OCR0.
70     OCR0 = 125; // Timer interrupt will be generated when TCNT0==OCR0
71     // We want a 1 ms tick. 0.001 s * 125,000 ticks/s = 125
72     // So when TCNT0 register equals 125,
73     // 1 ms has passed. Thus, we compare to 125.

```

Apr 29, 13 20:59

jdefo002_lab6_part3.c

Page 2/4

```

71     // AVR timer interrupt mask register
72     TIMSK = 0x02; // bit1: OCIE0 -- enables compare match interrupt
73
74     //Initialize avr counter
75     TCNT0=0;
76
77     _avr_timer_cntcurr = _avr_timer_M;
78     // TimerISR will be called every _avr_timer_cntcurr milliseconds
79
80     //Enable global interrupts
81     SREG |= 0x80; // 0x80: 1000000
82 }
83
84 void TimerOff()
85 {
86     TCCR0 = 0x00; // bit2bit1bit0=000: timer off
87 }
88
89 void TimerISR()
90 {
91     TimerFlag = 1;
92 }
93
94 // In our approach, the C programmer does not touch this ISR, but rather TimerIS
95   R()
96   ISR(TIMER0_COMP_vect)
97   {
98       // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn se
99       ttings)
100       _avr_timer_cntcurr--; // Count down to 0 rather than up to TOP
101       if (_avr_timer_cntcurr == 0)
102       { // results in a more efficient compare
103           TimerISR(); // Call the ISR that the user uses
104           _avr_timer_cntcurr = _avr_timer_M;
105       }
106 }
107
108 // Set TimerISR() to tick every M ms
109 void TimerSet(unsigned long M)
110 {
111     _avr_timer_M = M;
112     _avr_timer_cntcurr = _avr_timer_M;
113 }
114
115 enum Counter_States { Init, ShiftLightAndCheck, WinState } CounterState;
116
117 unsigned char TckFct_Counter(unsigned char inputData, unsigned char LedValue)
118 {
119     // Variable we are returning
120     unsigned char tempLedValue = LedValue;
121
122     switch(CounterState)
123     {
124         //Transitions
125         case Init: // Initial Transition
126             CounterState = ShiftLightAndCheck;
127             break;
128         case ShiftLightAndCheck:
129             if(LedValue > 0x04)
130             {
131                 CounterState = Init;
132             }
133             else if((LedValue == 0x02) && (inputData == 0x01))
134             {
135                 CounterState = WinState;
136             }
137             else
138             {
139                 CounterState = ShiftLightAndCheck;
140             }
141             break;
142         case WinState:
143             if(inputData == 0x01)

```

Apr 29, 13 20:59

jdefo002_lab6_part3.c

Page 3/4

```

142         {
143             CounterState = Init;
144         }
145         else
146         {
147             CounterState = WinState;
148         }
149         break;
150     default:
151         break;
152 }
153
154 switch(CounterState)
155 { // Actions
156     case Init:
157         tempLedValue = 0x01;
158         break;
159     case ShiftLightAndCheck:
160         tempLedValue <= 1;
161         break;
162     case WinState:
163         tempLedValue = 0x02;
164         break;
165     default:
166         break;
167 }
168 return tempLedValue;
169 }
170
171 unsigned char Write7Seg(unsigned char x)
172 {
173     unsigned char segvalue = 0;
174
175     switch(x)
176     {
177         case 0:
178             segvalue = 0x3F;
179             break;
180         case 1:
181             segvalue = 0x0C;
182             break;
183         case 2:
184             segvalue = 0x5B;
185             break;
186         case 3:
187             segvalue = 0x5E;
188             break;
189         case 4:
190             segvalue = 0x6C;
191             break;
192         case 5:
193             segvalue = 0x76;
194             break;
195         case 6:
196             segvalue = 0x77;
197             break;
198         case 7:
199             segvalue = 0x1C;
200             break;
201         case 8:
202             segvalue = 0x7F;
203             break;
204         case 9:
205             segvalue = 0x7C;
206             break;
207         default:
208             segvalue = 0x73;
209             break;
210     }
211
212     return segvalue;
213 }
214

```

Apr 29, 13 20:59

jdefo002_lab6_part3.c

Page 4/4

```

215 int main(void)
216 {
217     SENSOR_DDR = 0x00;
218     LED_DDR    = 0xFF;
219     UNSEDC_DDR = 0x00;
220     UNSEDD_DDR = 0x00;
221
222     // Initialize LEDs to off
223     LED_OUTPORT = 0x00;
224     unsigned char ButtonValue = 0;
225     unsigned char CanAcceptButtonPressed = 0;
226     unsigned char CurrentLEDValue = 0x01;
227     unsigned char SevenSegValue = 0;
228
229     // The Timer is set to 500 ms because In the increment/decrement state i
230     t has // to go back to the wait for button press state. Therefore, there are t
231     wo // state transitions, each take up 500ms making a total of 1 second.
232     TimerSet(300);
233     TimerOn();
234     CounterState = Init;
235
236     CanAcceptButtonPressed = 1;
237     while(1)
238     {
239         // Wait for 1 second before advancing the state machine. While w
240         aiting see if // any buttons have been pressed. Save the last button pressed v
241         alues //ButtonValue = 0; // Assume no buttons have been pressed
242         TimerFlag = 0; // Reset the timer flag. It will be set by t
243         he timer0 interrupt.
244
245         // Wait until the timer0 interrupt executes (Once every second)
246         ButtonValue = 0;
247         while (!TimerFlag)
248         {
249             // If some buttons have been pressed then save what has
250             been pressed.
251             if ((~SENSOR_INPORT & 0x03) == 0)
252             {
253                 if (CanAcceptButtonPressed == 1)
254                     ButtonValue = 0;
255                 else
256                     CanAcceptButtonPressed = 1;
257             }
258             else
259             {
260                 if (CanAcceptButtonPressed == 1)
261                 {
262                     ButtonValue = (~SENSOR_INPORT & 0x03);
263                     CanAcceptButtonPressed = 0;
264                 }
265             }
266         }
267         if ((CounterState == WinState))
268             CanAcceptButtonPressed = 1;
269         // Now that 1 second is up we can advance the state machine
270         CurrentLEDValue = TckFct_Counter(ButtonValue, CurrentLEDValue);
271         // Write the reset to the seven segment display
272         //SevenSegValue = Write7Seg(CurrentLEDValue);
273         LED_OUTPORT = CurrentLEDValue;
274     }
275 }

```