```
/* jdefo002_lab6_part1.c − April 29, 2013
 * Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * CS Login: jdefo002
 * Partner Name: Ariana DeJaco Email adejaoo1@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#6 Exercise#1
 * Exercise Description: Create a synchSM to blink three LEDs
 * connected to PB0, PB1, and PB2 in sequence, 1 second each.
 * Implement that synchSM in C using the method defined in class.
 * In addition to demoing your program, you will need to show that
 * your code adheres entirely to the method with no variations.
 */

#include <avr/io.h>
#include <avr/interrupt.h>

// Current Port Definitions
#define LED_DDR                         DDRB
#define LED_INPORT                      PINB
#define LED_OUTPORT                     PORTB
#define SENSOR_DDR                      DDRA
#define SENSOR_INPORT           PINA
#define SENSOR_OUTPORT          PORTA
#define UNUSEDC_DDR         DDRC
#define UNUSEDC_PIN         PINC
#define UNUSEDC_PORT        PORTC
#define UNUSEDD_DDR         DDRD
#define UNUSEDD_PIN         PIND
#define UNUSEDD_PORT        PORTD

// Additional macros not defines in sfr_defs.h
#define SET_PORT_BIT(OUTPORT, BIT)              OUTPORT |= (1 << BIT)
#define CLEAR_PORT_BIT(OUTPORT, BIT)    OUTPORT &= ~(1 << BIT)

//DDRA: Configures each of port A's physical pins to input (0) or output(1)
//PORTA: Writing to this register writes the port's physical pins
// (Write only)
//PINA: Reading this register reads the values of the port's physical pins
// (Read only)

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

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

void TimerOn()
{
        // AVR timer/counter controller register TCCR0
        TCCR0 = 0x0B;   // bit3bit6=10: CTC mode (clear timer on compare)
        // bit2bit1bit0=011: pre−scaler /64
        // 00001011: 0x0B
        // SO, 8 MHz clock or 8,000,000 /64 = 125,000 ticks/s
        // Thus, TCNT0 register will count at 125,000 ticks/s

        // AVR output compare register OCR0.
        OCR0 = 125;     // Timer interrupt will be generated when TCNT0==OCR0
        // We want a 1 ms tick. 0.001 s * 125,000 ticks/s = 125
        // So when TCNT0 register equals 125,
        // 1 ms has passed. Thus, we compare to 125.
        // AVR timer interrupt mask register
        TIMSK = 0x02; // bit1: OCIE0 −− enables compare match interrupt

        //Initialize avr counter
        TCNT0=0;

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

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



```
}

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

void TimerISR()
{
        TimerFlag = 1;
}

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

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

enum BL_States { Initial, BL_LedOff, BL_LedOneOn, BL_LedTwoOn, BL_LedThreeOn } B
L_State;

void TickFct_Blink()
{
        switch ( BL_State )
        { //Transitions
                case Initial:
                        BL_State = BL_LedOff; //Initial state
                        break;
                case BL_LedOff:
                        BL_State = BL_LedOneOn;
                        break;
                case BL_LedOneOn:
                        BL_State = BL_LedTwoOn;
                        break;
                case BL_LedTwoOn:
                        BL_State = BL_LedThreeOn;
                        break;
                case BL_LedThreeOn:
                        BL_State = BL_LedOff;
                        break;
                default:
                        BL_State = −1;
                break;
        }

        switch (BL_State )
        { //State actions
                case BL_LedOff:
                        LED_OUTPORT = 0x00;
                        break;
                case BL_LedOneOn:
                        LED_OUTPORT = 0x01;
                        break;
                case BL_LedTwoOn:
                        LED_OUTPORT = 0x02;
                        break;
                case BL_LedThreeOn:
                        LED_OUTPORT = 0x04;
```

```
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 }
```

```
/* jdefo002_lab6_part2.c − April 29, 2013
 * Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * CS Login: jdefo002
 * Partner Name: Ariana DeJaco Email adeja001@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#6 Exercise#2
 * Exercise Description: (From an earlier lab) Buttons are connected
 * to PA0 and PA1. Output PORTC drives the 7−segment display initially
 * displaying 0.  Pressing PA0 increments the display (stopping at 9).
 * Pressing PA1 decrements the display (stopping at 0). If both buttons
 * are depressed (even if not initially simultaneously), the display
 * resets to 0. If a button is held, then the display continues to
 * increment (or decrement) at a rate of once per second. Use a
 * synchronous state machine captured in C.
 */

#define F_CPU 8000000
// Interrupt.h is needed otherwise it will NOT work!!!!

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/sfr_defs.h>

// Bit−access function
unsigned char GetBit(unsigned char x, unsigned char k)
{
        return ((x & (0x01 << k)) != 0);
}

// Current Port Definitions
#define LED_DDR                         DDRC
#define LED_INPORT                      PINC
#define LED_OUTPORT                     PORTC
#define SENSOR_DDR                      DDRA
#define SENSOR_INPORT           PINA
#define SENSOR_OUTPORT          PORTA
#define UNUSEDB_DDR         DDRB
#define UNUSEDB_PIN         PINB
#define UNUSEDB_PORT        PORTB
#define UNUSEDD_DDR         DDRD
#define UNUSEDD_PIN         PIND
#define UNUSEDD_PORT        PORTD

 // Additional macros not defines in sfr_defs.h
 #define SET_PORT_BIT(OUTPORT, BIT)             OUTPORT |= (1 << BIT)
 #define CLEAR_PORT_BIT(OUTPORT, BIT)   OUTPORT &= ~(1 << BIT)

//DDRA: Configures each of port A's physical pins to input (0) or output(1)
//PORTA: Writing to this register writes the port's physical pins
// (Write only)
//PINA: Reading this register reads the values of the port's physical pins
// (Read only)

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

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

void TimerOn()
{
        // AVR timer/counter controller register TCCR0
        TCCR0 = 0x0B;   // bit3bit6=10: CTC mode (clear timer on compare)
        // bit2bit1bit0=011: pre−scaler /64
        // 00001011: 0x0B
        // SO, 8 MHz clock or 8,000,000 /64 = 125,000 ticks/s
        // Thus, TCNT0 register will count at 125,000 ticks/s

        // AVR output compare register OCR0.
```



```
        OCR0 = 125;     // Timer interrupt will be generated when TCNT0==OCR0
        // We want a 1 ms tick. 0.001 s * 125,000 ticks/s = 125
        // So when TCNT0 register equals 125,
        // 1 ms has passed. Thus, we compare to 125.
        // AVR timer interrupt mask register
        TIMSK = 0x02; // bit1: OCIE0 −− enables compare match interrupt

        //Initialize avr counter
        TCNT0=0;

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

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

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

void TimerISR()
{
        TimerFlag = 1;
}

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

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

enum Counter_States { InitReset, WaitForButtonPress, Increment, Decrement, Error
State } CounterState;

unsigned char TckFct_Counter(unsigned char inputData, unsigned char LedValue)
{
        // Variable we are returning
        unsigned char tempLedValue = LedValue;
        unsigned char checkIncrement = 0;
        unsigned char checkDecrement = 0;
        unsigned char i = 0;

        switch(CounterState)
        {
                //Transitions
                case InitReset:  // Initial Transition
                         CounterState = WaitForButtonPress;
                         break;
                case WaitForButtonPress:  // Initial Transition
                        if(inputData == 0x00)
                        {
                                CounterState = WaitForButtonPress;
                        }
                        else if(inputData == 0x01)
                        {
                                CounterState = Increment;
```

```
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                         }
185                         break;
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                         }
201                         break;
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;
```



```
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                 deault:
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         UNUSEDB_DDR= 0x00;
265         UNUSEDD_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
    t has
275         // to go back to the wait for button press state. Therefore, there are t
    wo
276         // 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
    aiting see if
283                 // any buttons have been pressed. Save the last button pressed v
```

```
alues
                ButtonValue = 0;    // Assume no buttons have been pressed
                TimerFlag = 0;      // Reset the timer flag. It will be set by t
he timer0 interrupt.

                // Wait until the timer0 interrupt executes (Once every second)
                while (!TimerFlag)
                {
                        // If some buttons have been pressed then save what has
 been pressed.
                        if ((~SENSOR_INPORT & 0x03) != 0)
                        {
                              ButtonValue = (~SENSOR_INPORT & 0x03);
                        }
                }
                // Now that  1 second is up we can advance the state machine
                CurrentLEDValue = TckFct_Counter(ButtonValue, CurrentLEDValue);
                // Write the reset to the seven segment display
                SevenSegValue = Write7Seg(CurrentLEDValue);
                LED_OUTPORT = SevenSegValue;
    }
}
```

```c
/* jdefo002_lab6_part3.c − April 29, 2013
 * Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * CS Login: jdefo002
 * Partner Name: Ariana DeJaco Email adeja001@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#6 Exercise#3/challenge
 * Exercise Description: (Challenge) Create a simple game that
 * requires pressing a button on PA0 while the middle of three
 * LEDs on PB0, PB1, and PB2 is lit. The LEDs light for 300 ms
 * each in sequence. When the button is pressed, the currently
 * lit LED stays lit. Pressing the button again restarts the game.
 */

// Interrupt.h is needed otherwise it will NOT work!!!!

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/sfr_defs.h>

// Bit−access function
unsigned char GetBit(unsigned char x, unsigned char k)
{
        return ((x & (0x01 << k)) != 0);
}

// Current Port Definitions
#define LED_DDR                         DDRB
#define LED_INPORT                      PINB
#define LED_OUTPORT                     PORTB
#define SENSOR_DDR                      DDRA
#define SENSOR_INPORT           PINA
#define SENSOR_OUTPORT          PORTA
#define UNUSEDC_DDR         DDRC
#define UNUSEDC_PIN         PINC
#define UNUSEDC_PORT        PORTC
#define UNUSEDD_DDR         DDRD
#define UNUSEDD_PIN         PIND
#define UNUSEDD_PORT        PORTD

 // Additional macros not defines in sfr_defs.h
 #define SET_PORT_BIT(OUTPORT, BIT)             OUTPORT |= (1 << BIT)
 #define CLEAR_PORT_BIT(OUTPORT, BIT)   OUTPORT &= ~(1 << BIT)

//DDRA: Configures each of port A's physical pins to input (0) or output(1)
//PORTA: Writing to this register writes the port's physical pins
// (Write only)
//PINA: Reading this register reads the values of the port's physical pins
// (Read only)

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

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

void TimerOn()
{
        // AVR timer/counter controller register TCCR0
        TCCR0 = 0x0B;   // bit3bit6=10: CTC mode (clear timer on compare)
        // bit2bit1bit0=011: pre−scaler /64
        // 00001011: 0x0B
        // SO, 8 MHz clock or 8,000,000 /64 = 125,000 ticks/s
        // Thus, TCNT0 register will count at 125,000 ticks/s

        // AVR output compare register OCR0.
        OCR0 = 125;     // Timer interrupt will be generated when TCNT0==OCR0
        // We want a 1 ms tick. 0.001 s * 125,000 ticks/s = 125
        // So when TCNT0 register equals 125,
        // 1 ms has passed. Thus, we compare to 125.
        // AVR timer interrupt mask register
        TIMSK = 0x02; // bit1: OCIE0 −− enables compare match interrupt

        //Initialize avr counter
        TCNT0=0;

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

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

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

void TimerISR()
{
        TimerFlag = 1;
}

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

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

enum Counter_States { Init, ShiftLightAndCheck, WinState } CounterState;

unsigned char TckFct_Counter(unsigned char inputData, unsigned char LedValue)
{
        // Variable we are returning
        unsigned char tempLedValue = LedValue;

        switch(CounterState)
        {
                //Transitions
                case Init:  // Initial Transition
                        CounterState = ShiftLightAndCheck;
                        break;
                case ShiftLightAndCheck:
                        if(LedValue > 0x04)
                        {
                                CounterState = Init;
                        }
                        else if((LedValue == 0x02) && (inputData == 0x01))
                        {
                                CounterState = WinState;
                        }
                        else
                        {
                                CounterState = ShiftLightAndCheck;
                        }
                        break;
                case WinState:
                    if(inputData == 0x01)
```

```
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                     deault:
208                             segvalue = 0x73;
209                             break;
210             }
211
212             return segvalue;
213     }
214
```



```
215     int main(void)
216     {
217             SENSOR_DDR = 0x00;
218             LED_DDR    = 0xFF;
219             UNUSEDC_DDR= 0x00;
220             UNUSEDD_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
    t has
230             // to go back to the wait for button press state. Therefore, there are t
    wo
231             // 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
    aiting see if
240                     // any buttons have been pressed. Save the last button pressed v
    alues
241                     //ButtonValue = 0;    // Assume no buttons have been pressed
242                     TimerFlag = 0;      // Reset the timer flag. It will be set by t
    he timer0 interrupt.
243
244                     // Wait until the timer0 interrupt executes (Once every second)
245                     ButtonValue = 0;
246                     while (!TimerFlag)
247                     {
248                             // If some buttons have been pressed then save what has
     been pressed.
249                             if ((~SENSOR_INPORT & 0x03) == 0)
250                             {
251                                     if (CanAcceptButtonPressed == 1)
252                                         ButtonValue = 0;
253                                     else
254                                        CanAcceptButtonPressed = 1;
255                             }
256                             else
257                             {
258                                     if (CanAcceptButtonPressed == 1)
259                                     {
260                                        ButtonValue = (~SENSOR_INPORT & 0x03);
261                                         CanAcceptButtonPressed = 0;
262                                     }

263                             }
264                     }
265                     if ((CounterState == WinState))
266                        CanAcceptButtonPressed = 1;
267                     // Now that  1 second is up we can advance the state machine
268                     CurrentLEDValue = TckFct_Counter(ButtonValue, CurrentLEDValue);
269                     // Write the reset to the seven segment display
270                     //SevenSegValue = Write7Seg(CurrentLEDValue);
271                     LED_OUTPORT = CurrentLEDValue;
272         }
273     }
```