```
/*      jdefo002_lab7_part1.c − 4/29/13
 *      Name: Joshua DeForest−Williams Email: Jdefo002@ucr.edu
 *      CS Login: jdefo002
 *      Partner(s) Name: Ariana Dejaco E−mail: Adeja001@ucr.edu
 *      Lab Section:022
 *      Assignment: Lab #7  Exercise #1
 *      Exercise Description:Connect LEDs to PB0, PB1, PB2, and PB3. Light
 *      the three LEDs PB0, PB1, and PB2 in sequence for 1 second each.
 *      Meanwhile, blink PB3 on 1 second and off 1 second.
 *      I acknowledge all content contained herein, excluding template or example
 *      code, is my own original work.
 */



#include <avr/io.h>
#include <avr/interrupt.h>
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 ms.
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 TimerISR()
ISR(TIMER0_COMP_vect)
{
   // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn settings)
   _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;
}
volatile unsigned char B = 0x01;
enum Counter_States { InitState, SecState, ThirdState } CounterState;

unsigned char TckFct_Counter()
{
        // Variable we are returning
```



```


        switch(CounterState)
        {
                //Transitions
                case InitState:  // Initial Transition
         B=0x01;
                        CounterState = SecState;
         break;
                case SecState:  // Initial Transition
         B=0x02;
                        CounterState = ThirdState;
         break;
                case ThirdState:  // Initial Transition
         B=0x04;
                        CounterState = InitState;
         break;
                default:
                        break;
        }

        switch(CounterState)
        { // Actions
                case InitState:
                        break;
                default:
                        break;
        }
}
enum Blink_State { InitialState, Blink_two } BlinkState;

unsigned char TckFct_Blink()
{
        // Variable we are returning


        switch(BlinkState)
        {
                //Transitions
                case InitialState:  // Initial Transition
         B = B | 0x08;
         BlinkState = Blink_two;
                        break;
       case Blink_two:
          B = B & 0x07;
          BlinkState = InitialState;
                default:
                        break;
        }

        switch(CounterState)
        { // Actions
                case InitState:
                        break;
                default:
                        break;
        }
}
void main()
{
    DDRB = 0xFF; // Set port B to output
    PORTB = 0x00; // Init port B to 0s
    TimerSet(1000);
    TimerOn();
    CounterState = InitState;
    BlinkState = InitialState;
    while(1) {
        // User code
        TckFct_Counter();
        TckFct_Blink();
        PORTB = B;
       while (!TimerFlag);    // Wait 1 sec
        TimerFlag = 0;
```

```
143             // Note: For the above a better style would use a synchSM with TickSM()
144             // This example just illustrates the use of the ISR and flag
145         }
146 }
```

```
/*      jdefo002_lab7_part2.c − 4/29/13
 *      Name: Joshua DeForest−Williams Email: Jdefo002@ucr.edu
 *      CS Login: jdefo002
 *      Partner(s) Name: Ariana Dejaco E−mail: Adeja001@ucr.edu
 *      Lab Section:022
 *      Assignment: Lab #7  Exercise #2
 *      Exercise Description:Connect LEDs to PB0, PB1, PB2, and PB3. Light
 *      the three LEDs PB0, PB1, and PB2 in sequence for 1 second each.
 *      Meanwhile, blink PB3 on 1 second and off 1 second.
 *      EXTENSION: Modify the above example so the three LEDs light for
 *      300 ms, while PB3's LED still blinks 1 second on and 1 second off.
 *      I acknowledge all content contained herein, excluding template or exampl
e
 *      code, is my own original work.
 */



#include <avr/io.h>
#include <avr/interrupt.h>
volatile unsigned char TimerFlag=0; // TimerISR() sets this to 1. C programmer s
hould 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 ms.
unsigned long _avr_timer_cntcurr=0; // Current internal count of 1ms ticks
volatile unsigned char B = 0x01;
volatile unsigned char Blink = 0x08;
volatile unsigned char Count_Time = 3;
volatile unsigned char Blink_Time = 10;
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;
   Count_Time++;
   Blink_Time++;
}
// 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 setting
s)
   _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 { InitState, SecState, ThirdState } CounterState;

unsigned char TckFct_Counter()
{
        // Variable we are returning


        switch(CounterState)
        {
                //Transitions
                case InitState:  // Initial Transition
         B=0x01;
                        CounterState = SecState;
         break;
                case SecState:  // Initial Transition
         B=0x02;
                        CounterState = ThirdState;
         break;
                case ThirdState:  // Initial Transition
         B=0x04;
                        CounterState = InitState;
         break;
                default:
                        break;
        }

        switch(CounterState)
        { // Actions
                case InitState:
                        break;
                default:
                        break;
        }
}
enum Blink_State { InitialState, Blink_two } BlinkState;

unsigned char TckFct_Blink()
{
        // Variable we are returning


        switch(BlinkState)
        {
                //Transitions
                case InitialState:  // Initial Transition
         Blink =0x08;
         BlinkState = Blink_two;
                        break;
      case Blink_two:
         Blink = 0x00;
         BlinkState = InitialState;
                default:
                        break;
        }

        switch(BlinkState)
        { // Actions
                case InitState:
                        break;
                default:
                        break;
        }
}
void main()
{
    DDRB = 0xFF; // Set port B to output
    PORTB = 0x00; // Init port B to 0s
    TimerSet(100);
    TimerOn();
    CounterState = InitState;
```

```
143     BlinkState = InitialState;
144     while(1) {
145         // User code
146         if (Count_Time == 3){
147             TckFct_Counter();
148             Count_Time = 0;
149         }
150         if (Blink_Time == 10){
151            TckFct_Blink();
152            Blink_Time = 0;
153         }
154         PORTB = B|Blink;
155        while (!TimerFlag);    // Wait 1 sec
156         TimerFlag = 0;
157         // Note: For the above a better style would use a synchSM with TickSM()
158         // This example just illustrates the use of the ISR and flag
159     }
160 }
```

```
/*      jdefo002_lab7_part3.c − 4/29/13
 *      Name: Joshua DeForest−Williams Email: Jdefo002@ucr.edu
 *      CS Login: jdefo002
 *      Partner(s) Name: Ariana Dejaco E−mail: Adeja001@ucr.edu
 *      Lab Section:022
 *      Assignment: Lab #7  Exercise #3
 *      Exercise Description:Connect LEDs to PB0, PB1, PB2, and PB3. Light
 *      the three LEDs PB0, PB1, and PB2 in sequence for 1 second each.
 *      Meanwhile, blink PB3 on 1 second and off 1 second.
 *      EXTENSION: Modify the above example so the three LEDs light for
 *      300 ms, while PB3's LED still blinks 1 second on and 1 second off.
 *      EXTENSION 2: To the previous exercise's implementation, connect your
 *      speaker's red wire to PB4 and black wire to ground. Add a third task
 *      that toggles PB4 on for 2 ms and off for 2 ms as long as a switch on
 *      PA2 is in the on position.
 *      I acknowledge all content contained herein, excluding template or example
 *      code, is my own original work.
 */

#include <avr/io.h>
#include <avr/interrupt.h>
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 ms.
unsigned long _avr_timer_cntcurr=0; // Current internal count of 1ms ticks
volatile unsigned char A;
volatile unsigned char B = 0x01;
volatile unsigned char Blink = 0x08;
volatile unsigned char Sound = 0x00;
volatile unsigned long Count_Time = 300;
volatile unsigned long Blink_Time = 1000;
volatile unsigned long Sound_time = 2;
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;
   Count_Time++;
   Blink_Time++;
   Sound_time++;
}
// In our approach, the C programmer does not touch this ISR, but rather TimerISR()
ISR(TIMER0_COMP_vect)
{
   // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn settings)
   _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 { InitState, SecState, ThirdState } CounterState;

void TckFct_Counter()
{
        switch(CounterState)
        {
                //Transitions
                case InitState:
         B=0x01;
                        CounterState = SecState;
         break;
                case SecState:
         B=0x02;
                        CounterState = ThirdState;
         break;
                case ThirdState:
         B=0x04;
                        CounterState = InitState;
         break;
                default:
                        break;
        }
        switch(CounterState)
        { // Actions
                case InitState:
                        break;
                default:
                        break;
        }
}
enum Blink_State { BInitialState, Blink_two } BlinkState;

void TckFct_Blink()
{
        switch(BlinkState)
        {
                //Transitions
                case BInitialState:  // Initial Transition
         Blink =0x08;
         BlinkState = Blink_two;
                        break;
      case Blink_two:
         Blink = 0x00;
         BlinkState = BInitialState;
                default:
                        break;
        }
        switch(BlinkState)
        { // Actions
                case BInitialState:
                        break;
                default:
                        break;
        }
}
enum Sound_State { SInitialState, Sound_two } SoundState;

void TckFct_Sound()
{
        switch(SoundState)
        {
                //Transitions
                case SInitialState:  // Initial Transition
        if (A)
```

```
143          {
144             Sound = 0x10;
145          }
146          else{
147             Sound = 0x00;
148          }
149           SoundState = Sound_two;
150                      break;
151        case Sound_two:
152           Sound = 0x00;
153           SoundState = SInitialState;
154                  default:
155                      break;
156          }
157          switch(SoundState)
158          { // Actions
159                  case SInitialState:
160                      break;
161                  default:
162                      break;
163          }
164  }
165  int main(void)
166  {
167      DDRA = 0x00;
168      PORTA = 0xFF;
169      DDRB = 0xFF; // Set port B to output
170      PORTB = 0x00; // Init port B to 0s
171      TimerSet(1);
172      TimerOn();
173      CounterState = InitState;
174      BlinkState = BInitialState;
175      SoundState = SInitialState;
176      while(1) {
177          // User code
178          A = PINA;
179          A = ((A&0x04)>>2);
180          if (Count_Time == 300){
181              TckFct_Counter();
182              Count_Time = 0;
183          }
184          if (Blink_Time == 1000){
185             TckFct_Blink();
186             Blink_Time = 0;
187          }
188          if (Sound_time == 2)
189          {
190             TckFct_Sound();
191             Sound_time = 0;
192          }
193          PORTB = B|Blink|Sound;
194         //while (!TimerFlag);
195          TimerFlag = 0;
196          // Note: For the above a better style would use a synchSM with TickSM()
197          // This example just illustrates the use of the ISR and flag
198      }
199  }
```

```
/*      jdefo002_lab7_part4.c − 4/29/13
 *      Name: Joshua DeForest−Williams Email: Jdefo002@ucr.edu
 *      CS Login: jdefo002
 *      Partner(s) Name: Ariana Dejaco E−mail: Adeja001@ucr.edu
 *      Lab Section:022
 *      Assignment: Lab #7  Exercise #4
 *      Exercise Description:Connect LEDs to PB0, PB1, PB2, and PB3. Light
 *      the three LEDs PB0, PB1, and PB2 in sequence for 1 second each.
 *      Meanwhile, blink PB3 on 1 second and off 1 second.
 *      EXTENSION: Modify the above example so the three LEDs light for
 *      300 ms, while PB3's LED still blinks 1 second on and 1 second off.
 *      EXTENSION 2: To the previous exercise's implementation, connect your
 *      speaker's red wire to PB4 and black wire to ground. Add a third task
 *      that toggles PB4 on for 2 ms and off for 2 ms as long as a switch on
 *      PA2 is in the on position.
 *      I acknowledge all content contained herein, excluding template or example
 *      code, is my own original work.
 */

#include <avr/io.h>
#include <avr/interrupt.h>
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 ms.
unsigned long _avr_timer_cntcurr=0; // Current internal count of 1ms ticks
volatile unsigned char A;
volatile unsigned char B = 0x01;
volatile unsigned char Blink = 0x08;
volatile unsigned char Sound = 0x00;
volatile unsigned long Sound_var = 1;
volatile unsigned long Count_Time = 300;
volatile unsigned long Blink_Time = 1000;
volatile unsigned long Sound_time = 1;
volatile unsigned long button_press = 20;
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;
   Count_Time++;
   Blink_Time++;
   Sound_time++;
   button_press++;
}
// In our approach, the C programmer does not touch this ISR, but rather TimerISR()
ISR(TIMER0_COMP_vect)
{
   // CPU automatically calls when TCNT0 == OCR0 (every 1 ms per TimerOn settings)
```



```
    _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 { InitState, SecState, ThirdState } CounterState;

void TckFct_Counter()
{
        switch(CounterState)
        {
                //Transitions
                case InitState:
         B=0x01;
                        CounterState = SecState;
         break;
                case SecState:
         B=0x02;
                        CounterState = ThirdState;
         break;
                case ThirdState:
         B=0x04;
                        CounterState = InitState;
         break;
                default:
                        break;
        }
        switch(CounterState)
        { // Actions
                case InitState:
                        break;
                default:
                        break;
        }
}
enum Blink_State { BInitialState, Blink_two } BlinkState;

void TckFct_Blink()
{
        switch(BlinkState)
        {
                //Transitions
                case BInitialState:  // Initial Transition
         Blink =0x08;
         BlinkState = Blink_two;
                        break;
      case Blink_two:
         Blink = 0x00;
         BlinkState = BInitialState;
                default:
                        break;
        }
        switch(BlinkState)
        { // Actions
                case BInitialState:
                        break;
                default:
                        break;
        }
}
enum Sound_State { SInitialState, Sound_two } SoundState;

void TckFct_Sound()
{
        switch(SoundState)
        {
```

```
143                     //Transitions
144                     case SInitialState:  // Initial Transition
145             if (!A)
146             {
147                Sound = 0x30;
148             }
149             else{
150                Sound = 0x00;
151             }
152              SoundState = Sound_two;
153                             break;
154           case Sound_two:
155              Sound = 0x00;
156              SoundState = SInitialState;
157                     default:
158                             break;
159             }
160             switch(SoundState)
161             { // Actions
162                     case SInitialState:
163                             break;
164                     default:
165                             break;
166             }
167     }
168     enum Frequency_State { FInitialState} FrequencyState;
169
170     void TckFct_Freq()
171     {
172             switch(FrequencyState)
173             {
174           case FInitialState:
175           {
176              if (A & 0x01)
177              {
178                 Sound_var++;
179              }
180              else if ((A & 0x02)&& Sound_var > 1){
181                 Sound_var−−;
182              }
183              FrequencyState = FInitialState;
184              break;
185           }
186           default:
187                     break;
188                     }
189                     switch(SoundState)
190                     { // Actions
191                     case SInitialState:
192                     break;
193                     default:
194                     break;
195                     }
196     }
197     int main(void)
198     {
199         DDRA = 0x00;PORTA = 0xFF;
200
201         DDRB = 0xFF; // Set port B to output
202         PORTB = 0x00; // Init port B to 0s
203         TimerSet(1);
204         TimerOn();
205         CounterState = InitState;
206         BlinkState = BInitialState;
207         SoundState = SInitialState;
208         FrequencyState = FInitialState;
209         while(1) {
210             // User code
211             A = ~PINA;
212             if (Count_Time >= 300){
213                 TckFct_Counter();
214                 Count_Time = 0;
215             }
```



```
216             if (Blink_Time >= 1000){
217                TckFct_Blink();
218                Blink_Time = 0;
219             }
220             if (button_press >= 20){
221                TckFct_Freq ();
222                button_press = 0;
223                }
224             if (Sound_time >= Sound_var)
225             {
226                TckFct_Sound();
227                Sound_time = 0;
228             }
229             PORTB = B|Blink|Sound;
230            while (!TimerFlag);
231             TimerFlag = 0;
232             // Note: For the above a better style would use a synchSM with TickSM()
233             // This example just illustrates the use of the ISR and flag
234         }
235     }
```