```
/*
 * adeja001_lab3_part1.c − April 15, 2013
 * Name: Ariana DeJaco  E−mail:adeja001@ucr.edu
 * CS Login: adeja001
 * Partner Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#3 Exercise#1
 * Exercise Description: PB0 and PB1 each connect to an LED, and PB0's
 * LED is initially on. Pressing a button connected to PGetBit (INPUT_INPORT,0) turns off
 * PB0's LED and turns on PB1's LED, staying that way after button release.
 * Pressing the button again turns off PB1's LED and turns on PB0's LED.
*/


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


// Current Port Definitions
#define INPUT_DDR                       DDRA
#define INPUT_INPORT            PINA
#define INPUT_OUTPORT   PORTA
#define LED_DDR                 DDRB
#define LED_INPORT              PINB
#define LED_OUTPORT             PORTB

// 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)

 unsigned char flag = 0;
unsigned char GetBit(unsigned char x, unsigned char k) {
   return ((x & (0x01 << k)) != 0);
}
//volatile unsigned char TimerFlag=0; //raised by ISD, lowered by main code.
//void TimerISR() {
  // TimerFlag=1;
//}
enum LED_States{Init, LED0,LED1, reset_flag_led1, reset_flag_led0}
      LED_State;

void TickFct_LED(){
   switch (LED_State){ // Transitions
      case Init:
         LED_State = LED0; //Initial State
         break;
      case LED0:
         if (GetBit (INPUT_INPORT,0) && !flag){ // a=0
            LED_State = LED0;
         }
         else if(!GetBit (INPUT_INPORT,0)){ //a =1
             LED_State = reset_flag_led0;
         }

         break;
      case LED1:

         if (GetBit (INPUT_INPORT,0) && flag) // a=1
         {
            LED_State = LED1;
         }
/*         else if (!flag)
         {
            LED_State= LED0;
         }*/
         else if (!GetBit (INPUT_INPORT,0)) //a =0
         {
            LED_State = reset_flag_led1;
         }

         break;
      case reset_flag_led1:
```



```
            if (GetBit (INPUT_INPORT,0))
            {
               LED_State = LED0;
            }
            break;
       case reset_flag_led0:
         if (GetBit (INPUT_INPORT, 0))
         {
            LED_State = LED1;
         }
         break;
       default:
            LED_State =Init;
            break;
   }
   switch (LED_State){ //State Actions
      case LED0:
         SET_PORT_BIT(LED_OUTPORT,0);
         CLEAR_PORT_BIT(LED_OUTPORT,1);
         flag = 0;
         break;
      case LED1:
         SET_PORT_BIT(LED_OUTPORT,1);
         CLEAR_PORT_BIT(LED_OUTPORT,0);
         flag = 1;
         break;
      case reset_flag_led1:
         flag = 0;
         break;
      case reset_flag_led0:
         flag = 1;
         break;
      default:
         break;

   }
}

//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)
int main(void)
{
        LED_DDR = 0xFF; //Configures port B's 8 pins as outputs.
        LED_OUTPORT = 0x00; //Initialize output on port C to 0x00;
        INPUT_DDR = 0x00; // Configure Port A's 8 pins as inputs.
        INPUT_OUTPORT = 0xFF; // Configure Port A's 8 pins as inputs.
   LED_OUTPORT=0;
   //TimerSet(500);
   //TimerOn();
   LED_State = Init;
   while(1)
   {
     TickFct_LED ();
     //while(!TimerFlag) {}
        //TimerFlag = 0;
   }

}
```

```
/*
 * adeja001_lab3_part2.c − April 15, 2013
 * Name: Ariana DeJaco  E−mail:adeja001@ucr.edu
 * CS Login: adeja001
 * Partner Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#3 Exercise#2
 * Exercise Description: Buttons are connected to PA0 and PA1.
 * Output for PORTC is initially 0. Pressing PA0 increments PORTC
 * (stopping at 9). Pressing PA1 decrements PORTC (stopping at 0).
 * If both buttons are depressed (even if not initially simultaneously),
 * PORTC resets to 0.
 */


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


// Current Port Definitions
#define INPUT_DDR               DDRA
#define INPUT_INPORT            PINA
#define INPUT_OUTPORT           PORTA
#define COUNT_DDR               DDRC
#define COUNT_INPORT            PINC
#define COUNT_OUTPORT           PORTC

// 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)

unsigned char count = 0;
 unsigned char flag = 0;
unsigned char GetBit(unsigned char x, unsigned char k) {
   return ((x & (0x01 << k)) != 0);
}

enum COUNT_States{Init, Increment ,Decrement, waiting, both}
      COUNT_State;

void TickFct_CNT(){
   switch (COUNT_State){ // Transitions
      case Init:
         if (GetBit (INPUT_INPORT,0) && count < 9){
            COUNT_State = Increment; //Initial State
         }
         else if (GetBit (INPUT_INPORT,1)&& count > 0)
         {
            COUNT_State= Decrement;
         }
         else
         {
            COUNT_State = Init;
         }
         break;
      case Increment:
         if (GetBit (INPUT_INPORT,0)&& GetBit (INPUT_INPORT,1))
         {
            COUNT_State = both;
         }
                else if (GetBit(INPUT_INPORT,0) && !flag)
                {
                        COUNT_State = Increment;
                }
         else
         {
             COUNT_State = waiting;
         }

         break;
      case Decrement:
         if (GetBit (INPUT_INPORT,0)&& GetBit (INPUT_INPORT,1))
         {
```



```
            COUNT_State = both;
         }
                else if (GetBit(INPUT_INPORT,1) && !flag)
                {
                        COUNT_State = Decrement;
                }
         else
         {
            COUNT_State = waiting;
         }

         break;

       case waiting:
         if (GetBit (INPUT_INPORT,0)&& GetBit (INPUT_INPORT,1))
         {
            COUNT_State = both;
         }

         if (GetBit (INPUT_INPORT, 0))
         {
            COUNT_State = waiting;
         }
         else if (GetBit (INPUT_INPORT,1))
         {
            COUNT_State = waiting;
         }
         else if (!GetBit (INPUT_INPORT,0) && !GetBit (INPUT_INPORT,1))
         {
            COUNT_State = Init;
         }
         break;
       case both:
         COUNT_State = both;
         break;
       default:
         COUNT_State =Init;
         break;
    }
    switch (COUNT_State){ //State Actions
       case Increment:
         count = count +1;
         COUNT_State = waiting;
                flag = 1;
         break;
       case Decrement:
         count = count − 1;
         COUNT_State= waiting;
         flag = 1;
         break;
       case waiting:
         COUNT_State = waiting;
                flag = 0;
         break;
       case both:
         count = 0;
                flag = 0;
         COUNT_State = Init;
         break;
       default:
         break;

    }
    }


//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)
int main(void)
{
        COUNT_DDR = 0xFF; //Configures port B's 8 pins as outputs.
```

```
146           COUNT_OUTPORT = 0x00; //Initialize output on port C to 0x00;
147           INPUT_DDR = 0x00; // Configure Port A's 8 pins as inputs.
148           INPUT_OUTPORT = 0xFF; // Configure Port A's 8 pins as inputs.
149
150       COUNT_State = Init;
151       while(1)
152       {
153         TickFct_CNT ();
154           COUNT_OUTPORT = count;
155       }
156
157   }
```

```
/*
 * adeja001_lab3_part3.c − April 15, 2013
 * Name: Ariana Dejaco  E−mail:adeja001@ucr.edu
 * CS Login: Adeja001
 * Partner Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#3 Exercise#2
 * Exercise Description:A household has a digital combination deadbolt
 * lock system on the doorway. The system has buttons on a keypad. Button
 * 'X' connects to PA0, 'Y' to PA1, and '#' to PA2. Pressing and
 * releasing '#', then pressing 'Y', should unlock the door by setting
 * PB0 to 1. Any other sequence fails to unlock. Pressing a button from
 * inside the house (PA7) locks the door (PB0=0). For debugging purposes,
 * give each state a number, and always write the current state to PORTC
 * (consider using the enum state variable). Also, be sure to check that
 * only one button is pressed at a time.
 */
#include <avr/io.h>
#include <avr/sfr_defs.h>
// Current Port Definitions
#define INPUT_DDR               DDRA
#define INPUT_INPORT            PINA
#define INPUT_OUTPORT           PORTA
#define DOOR_DDR                DDRB
#define DOOR_INPORT             PINB
#define DOOR_OUTPORT            PORTB
#define DEBUG_DDR               DDRC
#define DEBUG_INPORT            PINC
#define DEBUG_OUTPORT           PORTC
// 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)
unsigned char GetBit(unsigned char x_bit, unsigned char k) {
   return ((x_bit & (0x01 << k)) != 0);
}
char state_num= 0;
char x;
char y;
char pound;
char lock;
char flag = 0;
enum Door_States{Initial,button_p, button_Pound_wait, wait_y, Release, button_Y,
 button_lock} Door_State;
void TickFct_DOOR(){
   switch (Door_State){ // Transitions
      case Initial: // state 0
                if ((x&&y)||(x&&pound)||(x&&lock)||(y&&pound)||(y&&lock)||(pound
&&lock) ){
                        Door_State = Release;
                }
                else if (pound&&!y)
                {
                        Door_State = button_p;
                }
                else if (lock){
                        Door_State = button_lock;
                }
        break;
          case button_p: // state 1
                if (!pound && y) {
                        Door_State = button_Y;
                }
                else if (!pound && !y) {
                        Door_State = wait_y;
                }

                else
                {
                        Door_State = button_Pound_wait;
                }

                break;
       case button_Pound_wait:  // state 2
```



```
                if ((x&&y)||(x&&pound)||(x&&lock)||(y&&pound)||(y&&lock)||(pound
&&lock) ){
                        Door_State = Release;
                }
                else if (pound && flag)
                {
                        Door_State = button_Pound_wait;
                }
                else if (!pound && y) {
                        Door_State = button_Y;
                }
                else if (!pound && !y) {
                        Door_State = wait_y;
                }
        break;
          case wait_y: // state 3
                if ((pound && !y) || (pound && y))
                {
                        Door_State = Initial;
                }
                else if (!pound && !y)
                {
                        Door_State = wait_y;
                }
                else if (!pound && y)
                {
                        Door_State = button_Y;
                }
       case Release: // state 4
                if ((x&&y)||(x&&pound)||(x&&lock)||(y&&pound)||(y&&lock)||(pound
&&lock) ){
                        Door_State = Release;
                }
                else
                {
                        Door_State = Initial;
                }
          break;
       case button_Y:// state 5
                 Door_State = Initial;
          break;
       case button_lock:// state 6
          break;
       default:

          break;

    }

    switch (Door_State){ //State Actions
            case button_p:
                 Door_State = button_Pound_wait;
                 flag = 1;
                 state_num = 1;
                 break;
       case button_Pound_wait:
                  state_num = 2;
          break;
           case wait_y:
                  flag = 0;
                  state_num = 3;
                  break;
       case Release:
                   state_num = 4;
          break;
       case button_Y:
                  SET_PORT_BIT(DOOR_OUTPORT, 0);
                  flag = 0;
                  state_num = 5;
          break;
       case button_lock:
                  CLEAR_PORT_BIT(DOOR_OUTPORT,0);
                  Door_State = Initial;
```

```
143                     state_num = 6;
144             break;
145         default:
146                     state_num= 0;
147             break;
148             }
149     }
150 //DDRA: Configures each of port A's physical pins to input (0) or output (1)
151 //PORTA: Writing to this register writes the port's physical pins (Write only)
152 //PINA: Reading this register reads the values of the port's physical pins (Read
     only)
153 int main(void)
154 {
155         DOOR_DDR = 0xFF; //Configures port B's 8 pins as outputs.
156         DOOR_OUTPORT = 0x00; //Initialize output on port B to 0x00;
157         DEBUG_DDR = 0xFF; //Configures port C's 8 pins as outputs.
158         DEBUG_OUTPORT = 0x00; //Initialize output on port C to 0x00;
159         INPUT_DDR = 0x00; // Configure Port A's 8 pins as inputs.
160         INPUT_OUTPORT = 0xFF; // Configure Port A's 8 pins as inputs.
161     Door_State = Initial;
162     while(1)
163     {
164           x = GetBit(INPUT_INPORT, 0);
165           y = GetBit(INPUT_INPORT, 1);
166           pound = GetBit(INPUT_INPORT, 2);
167           lock = GetBit(INPUT_INPORT, 7);
168       TickFct_DOOR ();
169       DEBUG_OUTPORT = state_num;
170     }
171 }
```

```
/*
 * adeja001_lab3_part4.c − April 15, 2013
 * Name: Ariana Dejaco  E−mail:adeja001@ucr.edu
 * CS Login: adeja001
 * Partner Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#3 Exercise#4
 * Exercise Description:A household has a digital combination deadbolt
 * lock system on the doorway. The system has buttons on a keypad. Button
 * 'X' connects to PA0, 'Y' to PA1, and '#' to PA2. Pressing and
 * releasing '#', then pressing 'Y', should unlock the door by setting
 * PB0 to 1. Any other sequence fails to unlock. Pressing a button from
 * inside the house (PA7) locks the door (PB0=0). For debugging purposes,
 * give each state a number, and always write the current state to PORTC
 * (consider using the enum state variable). Also, be sure to check that
 * only one button is pressed at a time.
 * (Challenge) Extend the above door so that it can also be locked by entering t
he earlier code.
 */
#include <avr/io.h>
#include <avr/sfr_defs.h>
// Current Port Definitions
#define INPUT_DDR               DDRA
#define INPUT_INPORT            PINA
#define INPUT_OUTPORT           PORTA
#define DOOR_DDR                DDRB
#define DOOR_INPORT             PINB
#define DOOR_OUTPORT            PORTB
#define DEBUG_DDR               DDRC
#define DEBUG_INPORT            PINC
#define DEBUG_OUTPORT           PORTC
// 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)
unsigned char GetBit(unsigned char x_bit, unsigned char k) {
   return ((x_bit & (0x01 << k)) != 0);
}
char state_num= 0;
char x;
char y;
char pound;
char lock;
char flag = 0;
char locked_flag = 0; // locked = 0, unlocked = 1;
enum Door_States{Initial,button_p, button_Pound_wait, wait_y, Release, button_Y,
 button_lock,
                locked_y_wait, locked_y} Door_State;
void TickFct_DOOR(){
   switch (Door_State){ // Transitions
      case Initial: // state 0
                if ((x&&y)||(x&&pound)||(x&&lock)||(y&&pound)||(y&&lock)||(pound
&&lock) ){
                        Door_State = Release;
                }
                else if (pound&&!y)
                {
                        Door_State = button_p;
                }
                else if (lock){
                        Door_State = button_lock;
                }
        break;
          case button_p: // state 1
                if (!pound && y) {
                        Door_State = button_Y;
                }
                else if (!pound && !y) {
                        Door_State = wait_y;
                }

                else
                {
                        Door_State = button_Pound_wait;
```



```
                }

                break;
       case button_Pound_wait:  // state 2
                if ((x&&y)||(x&&pound)||(x&&lock)||(y&&pound)||(y&&lock)||(pound
&&lock) ){
                        Door_State = Release;
                }
                else if (pound && flag && locked_flag)
                {
                        Door_State = locked_y_wait;
                }
                else if (pound && flag && !locked_flag)
                {
                        Door_State = button_Pound_wait;
                }
                else if (!pound && y) {
                        Door_State = button_Y;
                }
                else if (!pound && !y) {
                        Door_State = wait_y;
                }
        break;
          case wait_y: // state 3
                if ((pound && !y) || (pound && y))
                {
                        Door_State = Initial;
                }
                else if (!pound && !y)
                {
                        Door_State = wait_y;
                }
                else if (!pound && y)
                {
                        Door_State = button_Y;
                }
       case Release: // state 4
                if ((x&&y)||(x&&pound)||(x&&lock)||(y&&pound)||(y&&lock)||(pound
&&lock) ){
                        Door_State = Release;
                }
                else
                {
                        Door_State = Initial;
                }
          break;
       case button_Y:// state 5
                 Door_State = Initial;
          break;
       case button_lock:// state 6
          break;
           case locked_y_wait: // state 7
                 if ((pound && !y) || (pound && y))
                {
                        Door_State = Initial;
                }
                else if (!pound && !y)
                {
                        Door_State = locked_y_wait;
                }
                else if (!pound && y)
                {
                        Door_State = locked_y;
                }
                 break;
           case locked_y:
                 break;
           default:

          break;

    }
```

```
142     switch (Door_State){ //State Actions
143             case button_p:
144                  Door_State = button_Pound_wait;
145                  flag = 1;
146                  state_num = 1;
147                  break;
148        case button_Pound_wait:
149                   state_num = 2;
150           break;
151            case wait_y:
152                   flag = 0;
153                   state_num = 3;
154                   break;
155        case Release:
156                    state_num = 4;
157           break;
158        case button_Y:
159                   SET_PORT_BIT(DOOR_OUTPORT, 0);
160                   flag = 0;
161                   state_num = 5;
162                   locked_flag = 1;
163           break;
164        case button_lock:
165                   CLEAR_PORT_BIT(DOOR_OUTPORT,0);
166                   Door_State = Initial;
167                   state_num = 6;
168                   locked_flag = 0;
169           break;
170            case locked_y_wait:
171                   state_num = 7;
172                   Door_State = locked_y_wait;
173                   break;
174        case locked_y:
175                   state_num = 8;
176                   CLEAR_PORT_BIT(DOOR_OUTPORT, 0);
177                   locked_flag = 0;
178                   Door_State = Initial;
179                   break;
180            default:
181                   state_num= 0;
182           break;
183           }
184    }
185  //DDRA: Configures each of port A's physical pins to input (0) or output (1)
186  //PORTA: Writing to this register writes the port's physical pins (Write only)
187  //PINA: Reading this register reads the values of the port's physical pins (Read
      only)
188  int main(void)
189  {
190          DOOR_DDR = 0xFF; //Configures port B's 8 pins as outputs.
191          DOOR_OUTPORT = 0x00; //Initialize output on port B to 0x00;
192          DEBUG_DDR = 0xFF; //Configures port C's 8 pins as outputs.
193          DEBUG_OUTPORT = 0x00; //Initialize output on port C to 0x00;
194          INPUT_DDR = 0x00; // Configure Port A's 8 pins as inputs.
195          INPUT_OUTPORT = 0xFF; // Configure Port A's 8 pins as inputs.
196     Door_State = Initial;
197     while(1)
198     {
199           x = GetBit(INPUT_INPORT, 0);
200           y = GetBit(INPUT_INPORT, 1);
201           pound = GetBit(INPUT_INPORT, 2);
202           lock = GetBit(INPUT_INPORT, 7);
203       TickFct_DOOR ();
204       DEBUG_OUTPORT = state_num;
205     }
206  }
```