```
/* adeja001_lab4_part1.c − April 22, 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#4 Exercise#1
 * Exercise Description: A car has a fuel−level sensor that sets PA3..PA0 to a value between 0 (empty) and 15 (full).
 * A series of LEDs connected to PC5..PC0 should light to graphically indicate the fuel level. If the fuel level is 1 or 2,
 * PC5 lights. If the level is 3 or 4, PC5 and PC4 light. Level 5−6 lights PC5..PC3. 7−9 lights PC5..PC2. 10−12
 * lights PC5..PC1. 13−15 lights PC5..PC0. Also, PC6 connects to a "Low fuel" icon, which should light if the level is 4 or less.
 * (The example below shows the display for a fuel level of 3).
 */


#include <avr/io.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

// All "fuel levels" defined as macros, makes easy to change
#define LEVEL1 2
#define LEVEL2 4
#define LEVEL3 6
#define LEVEL4 9
#define LEVEL5 12

// 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)

// Function to set all the LED lights according to the Fuel Level.
unsigned char TickFct_gas(char SensorValue)
{
        unsigned char ledValue;

    if (SensorValue > LEVEL5)
        {
            ledValue = 0x3F;
        }
    else if (SensorValue > LEVEL4)
        {
            ledValue = 0x3E;
        }
    else if (SensorValue > LEVEL3)
        {
            ledValue = 0x3C;
        }
    else if (SensorValue > LEVEL2)
        {
            ledValue = 0x38;
        }
```



```
    else if (SensorValue > LEVEL1)
        {
            ledValue = 0x30;
        }
        else if (SensorValue > 0)
        {
            ledValue = 0x20;
        }
    else
        {
        ledValue = 0;
        }

        if (SensorValue <= LEVEL2)
        {
            ledValue |= 0x40;
        }
        return ledValue;
}

//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)
{
   SENSOR_DDR = 0x00;   // Configure port A's 8 pins as inputs
   LED_DDR = 0xFF;      // Configure port C's 8 pins as outputs,
   UNUSEDB_DDR = 0X00;
   UNUSEDD_DDR = 0X00;

        unsigned char SensorValue = 0;
        unsigned char SavedSensorValue = 0;

    while(1)
    {
            // We are working with inverted logic so we need to use the complimen
t so
            // we use the tilda.. Must also mask out bits we are not using
            // (in this case A4−A7)
            SensorValue = (~SENSOR_INPORT & 0x0F);

            // Do not need to use the tilda (compliment) when putting into the fu
nction
            // because we need it to output a 1 in order for LED to light up.
       LED_OUTPORT = TickFct_gas(SensorValue);
    }
}
```

```c
/* adeja001_lab4_part2.c − April 22, 2013
 * Name: Ariana DeJaco Email adeja001@ucr.edu
 * CS Login: adeja001
 * Partner Name: Joshua DeForest−Williams E−mail jdefo002@ucr.edu
 * Lab Section: 022
 * Assignment: Lab#4 Exercise#2
 * Exercise Description: Buttons are connected to PA0 and PA1. Output for PORTC is initially 0.
 * Pressing PA0 increments PORTC. Pressing PA1 decrements. If both are pressed, PORTC is reset.
 */

// Used for debouncing the button (How many Hz)
#define F_CPU 1000000

#include <avr/io.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)

enum Counter_States { InitReset, Increment, Decrement, WaitForButtonPress, WaitForButtonRelease, ErrorState } CounterState;

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

        switch(CounterState)
        {
                //Transitions
                case InitReset:  // Initial Transition
                        CounterState = WaitForButtonRelease;
                        break;
                case Increment:
                    CounterState = WaitForButtonRelease;
                        break;
                case Decrement:
                        CounterState = WaitForButtonRelease;
                        break;
                case WaitForButtonPress:
                    if(inputData == 0x00)
                        {
                                CounterState = WaitForButtonPress;
                        }
                        else if(inputData == 0x01)
                        {
                                if(tempLedValue < 9)
                                {
                                        CounterState = Increment;
                                }
                                // Will stay in wait if not < 9
                        }
                        else if(inputData == 0x02)
                        {
                                if(tempLedValue > 0)
                                {
                                        CounterState = Decrement;
                                }
                                // Will stay in wait if not > 0
                        }
                        else if(inputData == 0x03)
                        {
                                CounterState = InitReset;
                        }
                        else
                        {
                                CounterState = ErrorState;
                        }
                        break;
                case WaitForButtonRelease:
                    if(inputData == 0x00)
                        {
                                CounterState = WaitForButtonPress;
                        }
                        break;
                case ErrorState:
                        break;
                default:
                        CounterState = ErrorState;
                        break;
        }

        switch(CounterState)
        { // Actions
                case Increment:
                        tempLedValue++;
                        break;
                case Decrement:
                        tempLedValue−−;
                        break;
                case InitReset:
                        tempLedValue = 0x00;
                        break;
                case WaitForButtonPress:
                        break;
                case WaitForButtonRelease:
                        break;
                case ErrorState:
                        tempLedValue = 0xFF;
                        break;
                default:
                        break;
        }
        return tempLedValue;
}

int main(void)
{
        SENSOR_DDR = 0x00;
        LED_DDR    = 0xFF;
        UNUSEDB_DDR= 0x00;
        UNUSEDD_DDR= 0x00;

        // Initialize LEDs to off
        LED_OUTPORT = 0x00;
        unsigned char ButtonValue = 0;
```

```
144         unsigned char CurrentLEDValue = 0;
145
146     while(1)
147     {
148                 // Code to debounce the switch.
149                 while (ButtonValue != (~SENSOR_INPORT & 0x03))
150                 {
151                         _delay_ms(50);
152                         ButtonValue = (~SENSOR_INPORT & 0x03);
153                 }
154
155                 CurrentLEDValue = TckFct_Counter(ButtonValue, CurrentLEDValue);
156                 LED_OUTPORT = CurrentLEDValue;
157     }
158 }
```

```
/* adeja001_lab4_partchallenge.c − April 22, 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#4 Exercise#Challenge
 * Exercise Description: Create your own festive lights display with 6 LEDs connected to PC5−PC0
 * lighting in some attractive sequence. Pressing the button on PA0 changes the lights to the next
 * configuration in the sequence.
 */

// Use for debouncing the switch (How many Hz)
#define F_CPU 1000000

#include <avr/io.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)

// Creating a type "statetpe" and making 2 variables of this type and initializing them to InitReset
typedef enum { InitReset, Increment, Decrement, WaitForButtonPress, WaitForButtonRelease, ShiftLeft, ShiftRight, ErrorState } statetype;
statetype CurrentState  = InitReset;
statetype ActionState = InitReset;

unsigned char TckFct_Counter(unsigned char inputData, unsigned char LedValue)
{
        unsigned char TempLedValue = LedValue;

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



```
                        {
                                CurrentState = WaitForButtonRelease;
                                ActionState = Increment;
                        }
                        break;
                case Decrement:
                        if(inputData == 0x00)
                        {
                                CurrentState = WaitForButtonPress;
                        }
                        else
                        {
                                CurrentState = WaitForButtonRelease;
                                ActionState = Decrement;
                        }
                        break;
                case ShiftRight:
                        if(inputData == 0x00)
                        {
                                CurrentState = WaitForButtonPress;
                        }
                        else
                        {
                                CurrentState = WaitForButtonRelease;
                                ActionState = ShiftRight;
                        }
                        break;
                case ShiftLeft:
                        if(inputData == 0x00)
                        {
                                CurrentState = WaitForButtonPress;
                        }
                        else
                        {
                                CurrentState = WaitForButtonRelease;
                                ActionState = ShiftLeft;
                        }
                        break;
                case WaitForButtonPress:
                    if(inputData == 0x00)
                        {
                                CurrentState = WaitForButtonPress;
                        }
                        else
                        {
                                switch(ActionState)
                                {
                                        case InitReset:
                                                CurrentState = ShiftLeft;
                                                break;
                                        case ShiftLeft:
                                                CurrentState = ShiftRight;
                                                break;
                                        case ShiftRight:
                                                CurrentState = Increment;
                                                break;
                                        case Increment:
                                                CurrentState = Decrement;
                                                break;
                                        case Decrement:
                                                CurrentState = ShiftLeft;
                                                break;
                                        default:
                                                CurrentState = ErrorState;
                                                break;
                                }
                        }
                case WaitForButtonRelease:
                    if(inputData == 0x00)
                        {
                                CurrentState = WaitForButtonPress;
                        }
                        break;
```

```
143                     case ErrorState:
144                             break;
145                     default:
146                             CurrentState = ErrorState;
147                             break;
148             }
149
150             switch(CurrentState)
151             {
152                     case InitReset:
153                             TempLedValue = 0x00;
154                             break;
155                     case Decrement:
156                             TempLedValue = 0x3F;
157                             break;
158                     case Increment:
159                             TempLedValue = 0x00;
160                             break;
161                     case ShiftLeft:
162                             TempLedValue = 0x01;
163                             break;
164                     case ShiftRight:
165                             TempLedValue = 0x20;
166                             break;
167                     case WaitForButtonRelease: //If button pressed or button release
    d do the action state
168                     case WaitForButtonPress:
169                             switch(ActionState)
170                             {
171                                     case Increment:
172                                         if(TempLedValue < 0x3F)
173                                             {
174                                                     TempLedValue++;
175                                             }
176                                             else
177                                             {
178                                                     TempLedValue = 0x00;
179                                             }
180                                             break;
181                                     case Decrement:
182                                             if(TempLedValue > 0x00)
183                                             {
184                                                     TempLedValue--;
185                                             }
186                                             else
187                                             {
188                                                     TempLedValue = 0x3F;
189                                             }
190                                             break;
191                                     case InitReset:
192                                             TempLedValue = 0x00;
193                                             break;
194                                     case ShiftLeft:
195                                             if(TempLedValue < 0x20)
196                                             {
197                                                     TempLedValue <<= 1;
198                                             }
199                                             else
200                                             {
201                                                     TempLedValue = 0x01;

202                                             }
203                                             break;
204                                     case ShiftRight:
205                                             if(TempLedValue > 0x01)
206                                             {
207                                                     TempLedValue >>= 1;
208                                             }
209                                             else
210                                             {
211                                                     TempLedValue = 0x20;
212                                             }
213                                             break;
```



```
214                                     case WaitForButtonPress:
215                                             TempLedValue = 0xFF; // Indicate Error
216                                             break;
217                                     case WaitForButtonRelease:
218                                             TempLedValue = 0xFF; // Indicate Error
219                                             break;
220                                     case ErrorState:
221                                             TempLedValue = 0xFF; // Indicate Error
222                                             break;
223                                     default:
224                                             break;
225                             }
226                             break;
227             }
228             return TempLedValue;
229     }
230
231     int main(void)
232     {
233             SENSOR_DDR = 0x00;
234             LED_DDR    = 0xFF;
235             UNUSEDB_DDR= 0x00;
236             UNUSEDD_DDR= 0x00;
237
238             LED_OUTPORT = 0x00;
239             unsigned char ButtonValue = 0;
240             unsigned char CurrentLEDValue = 0;
241
242         while(1)
243         {
244                     // De-bounce the switch
245                     while (ButtonValue != (~SENSOR_INPORT & 0x01))
246                     {
247                             _delay_ms(50);
248                             ButtonValue = (~SENSOR_INPORT & 0x01);
249                     }
250
251                     CurrentLEDValue = TckFct_Counter(ButtonValue, CurrentLEDValue);
252                     LED_OUTPORT = CurrentLEDValue;
253
254                     // Delay for the blinking lights.
255                     _delay_ms(250);
256         }
257     }
```