```
/*************** ECE2049 DEMO CODE ******************/
/*************** 14 May 2018   **********************/
/***************************************************/

#include <msp430.h>

#include "peripherals.h"
#include "utils/debug_assert.h"
#include "lecture.h"

void swDelay(char numLoops);
void initLeds(void);
void setLeds(char x);
char Lecture_readLaunchpadButtons(void);

#define BUTTON_RIGHT 0x01
#define BUTTON_LEFT 0x02


// Main
void main(void)
{
    WDTCTL = WDTPW | WDTHOLD;    // Stop watchdog timer. Always need to stop
this!!
                                 // You can then configure it properly, if
desired

    // *** System initialization ***
    initLaunchpadButtons();
    initLaunchpadLeds();
    configDisplay();
    initLeds(); // Configure LEDs

    while (1)    // Forever loop
    {
        // Input:  Read state of pins 1-0 from input register
        char inbits = Lecture_readLaunchpadButtons();

        // Set the LEDs to match our decoder specification
        switch(inbits)
        {
        case 0:
            setLeds(0x08); // 0000 1000b
            break;
        case 1:
            setLeds(0x04); // 0000 0100b
            break;
        case 2:
```

```c
            setLeds(0x02); // 0000 0010b
            break;
        case 3:
            setLeds(0x01); // 0000 0001b
            break;
        }

    }  // end while (1)
}

char Lecture_readLaunchpadButtons(void)
{
    // Output should be a bit vector with the state of
    // each pin in the lower two bits, as follows:
    // Bit     7  6  5  4  3  2    1    0
    // Output  0  0  0  0  0  0 P2.1 P1.1

    // Strategy:  read input register for each pin
    // Here, we shift each bit such taht each variable is either 0 or 1
    // (This is a generic strategy for working with inputs on various
    // pins--there are more efficient ways to accomplish this task)
    char b0 = (~P1IN & BIT1) >> 1;
    char b1 = (~P2IN & BIT1) >> 1;

    // Finally, we combine the variables to create the output
    // in the format we specified
    return (b1 << 1) | b0;
}


void initLeds(void)
{
    // Configure our four LEDs, on pins P6.3-0
    P6SEL &= ~(BIT3|BIT2|BIT1|BIT0); // Set LED pins for digital I/O
    P6DIR |= (BIT3|BIT2|BIT1|BIT0); // Set LED pins as outputs

    P6OUT &= ~(BIT3|BIT2|BIT1|BIT0); // Turn LEDs off (set output register to 0)
}

void setLeds(char x)
{
    // Given an input variable x, set
    // the LEDs according to the lower 4 bits of x
    // such that:
    // x 0000   x3   x2   x1   x0
    //        P6.0 P6.2 P6.1 P6.3

    // For each LED, we test the corresponding bit in
    // x and set it accordingly.
    // (We could do this more efficiently since all of
    // the pins are in a contiguous group.  This solution
    // is generic for any ordering of pins)

    if(x & BIT0) { // x0
        P6OUT |= BIT3;  // Set P6.0 to 1
    } else {
        P6OUT &= ~BIT3; // Set P6.0 to 0
    }

    if (x & BIT1) { // x1
        P6OUT |= BIT2;
    } else {
        P6OUT &= ~BIT2;
    }

    if (x & BIT2) { // x2
        P6OUT |= BIT1;
    } else {
        P6OUT &= ~BIT1;
    }

    if (x & BIT3) { // x3
        P6OUT |= BIT0;
    } else {
        P6OUT &= ~BIT0;
    }
}
```

```c
















void swDelay(char numLoops)
{
    // This function is a software delay. It performs
    // useless loops to waste a bit of time
    //
    // Input: numLoops = number of delay loops to execute
    // Output: none
    //
    // smj, ECE2049, 25 Aug 2013

    volatile unsigned int i,j;  // volatile to prevent removal in optimization
                                // by compiler. Functionally this is useless code

    for (j=0; j<numLoops; j++)
    {
        i = 20000 ;                 // SW Delay
        while (i > 0)               // could also have used while (i)
           i--;
    }
}

```