```c
/******************************************************************************
MSP430G2553 Project Creator

SE 423  - Jiaming Zhang
        Spring(2019)

        Written(by) : Steve(Keres)
College of Engineering Control Systems Lab
University of Illinois at Urbana-Champaign
*******************************************************************************/

#include "msp430g2553.h"
#include "UART.h"

void print_every(int rate);

char newprint = 0;
long NumOn = 0;
long NumOff = 0;
int statevar = 1;
int timecheck = 0;
char switch_state = '0';
//long B = (532 & 205);
//long C1 = (0x4f & 0x1ad);
//long D = 0x02ad | 0x1a1;
//long E = 0x3ba >> 4;
//long F = 104 << 3;
//long G = 495 & (0x5 << 4);
int Blink = 1;

char get_switchstate(void){

    if ((P2IN & 0xc0) == 0xc0) //no switch pressed
    {
        switch_state = '0';
    }
    else if ((P2IN & 0x80) == 0x80) //switch P2.6 is pressed
    {
        switch_state = '1';
    }
    else if ((P2IN & 0x40) == 0x40)  //switch P2.7 is pressed
    {
        switch_state = '2';
    }
    else if ((P2IN & 0x00) == 0)  //both switches are pressed
    {
        switch_state = '3';
    }

    return(switch_state);
}

void main(void) {

        WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT

        if (CALBC1_16MHZ ==0xFF || CALDCO_16MHZ == 0xFF) while(1);

        DCOCTL = CALDCO_16MHZ;    // Set uC to run at approximately 16 Mhz
        BCSCTL1 = CALBC1_16MHZ;

        // Initialize Port 1
        P1SEL &= ~0x01;  // See page 42 and 43 of the G2553's datasheet, It shows that when both P1SEL and
P1SEL2 bits are zero
        P1SEL2 &= ~0x01; // the corresponding pin is set as a I/O pin.  Datasheet:
http://coecsl.ece.illinois.edu/ge423/datasheets/MSP430Ref_Guides/msp430g2553datasheet.pdf
        P1REN = 0x0;  // No resistors enabled for Port 1
        P1DIR |= 0xf1; // Set P1.0 to output to drive LED on LaunchPad board.  Make sure shunt jumper is in
place at LaunchPad's Red LED
        P1OUT &= ~0x01;  // Initially set P1.0 to 0
        // Initialize Port 2
        P2SEL = 0x00;  // When both P1SEL and P1SEL2 bits are zero
        P2SEL2 = 0x00; // the corresponding pin is set as a general I/O pin.
        P2REN = 0xc0;  // P2.6 & P2.7 are enabled
        P2DIR = 0; // No output signals
        P2OUT = 0xc0;  // Initially set P2.0 to 11000000 to read signal from switch at P2.6 & P2.7



        // Timer A Config
        TACCTL0 = CCIE;                 // Enable Periodic interrupt
        TACCR0 = 16000;                // period = 1ms
        TACTL = TASSEL_2 + MC_1; // source SMCLK, up mode
        Init_UART(115200,1);    // Initialize UART for 115200 baud serial communication

        _BIS_SR(GIE);           // Enable global interrupt


    while(1) {  // Low priority Slow computation items go inside this while loop.  Very few (if anyt) items in the HWs will go inside this while loop

// for use if you want to use a method of receiving a string of chars over the UART see USCI0RX_ISR below
//      if(newmsg) {
//          newmsg = 0;
//      }

        // The newprint variable is set to 1 inside the function "print_every(rate)" at the given rate
        if ( (newprint == 1) && (senddone == 1) )  { // senddone is set to 1 after UART transmission is complete

            // only one UART_printf can be called every 15ms
            UART_printf("St%d On %ld Off %ld\n\r",statevar,NumOn,NumOff);

            newprint = 0;





            if (get_switchstate() == '0') //LED P1.4 on
            {
                UART_printf("LED P1.4 on ));
            }
            else if (get_switchstate() == '1')//LED P1.5 on
            {
                UART_printf("LED P1.5 on");
            }
            else if (get_switchstate() == '2')//LED P1.6 on
            {
                UART_printf("LED P1.6 on");
            }
            else if (get_switchstate() == '3')//LED P1.7 on
            {
                UART_printf("LED P1.7 on");
            }

//            if (cascadingprint == 1)
//            {
//                UART_printf("St%d On %ld Off %ld\n\r",statvar,NumOn,NumOff);
//            }
//            if (cascadingprint == 2)
//            {
//                UART_printf("%ld %ld %ld %ld %ld %ld \n\r", B,C1,D,E,F,G);
//            }



        }

    }
}


// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
    timecheck++; // Keep track of time for main while loop.
    print_every(500);  // units determined by the rate Timer_A ISR is called, print every "rate" calls to this function

//    switch (statevar) {
//        case 1:  //LED1 ON
//
//            P1OUT = 0x10;  // turn LED P1.4
//
//            if (timecheck == 500) {  // timecheck conut to 500 than turn of the LED and timecheck reset to 0,
//                timecheck = 0;
//                P1OUT = 0;
//                statevar = 2;  // Next Timer_A call go to state 2
//            } else {
//                statevar = 1;  // stays the same.  So not really needed
//            }
//            break;
//
//        case 2:  //LED2 ON
```

```
//
//            P1OUT = 0x20;  //turn LED P1.5
//
//            if (timecheck == 500) {  // if statement to determine what the state should be the next
millisecond into the Timer_A function
//                timecheck = 0;
//                P1OUT = 0;
//                statevar = 3;  // Next Timer_A call go to state 3
//            } else {
//                statevar = 2;  // stays the same.  So not really needed
//            }
//            break;
//
//        case 3:  //LED3 ON
//
//            P1OUT = 0x40;  // turn LED P1.6
//
//            if (timecheck == 500) {  // if statement to determine what the state should be the next
millisecond into the Timer_A function
//                timecheck = 0;
//                P1OUT = 0;
//                statevar = 4;  // Next Timer_A call go to state 4
//            } else {
//                statevar = 3;  // stays the same.  So not really needed
//            }
//            break;
//
//        case 4:  //LED4 ON
//
//            P1OUT = 0x80;  // turn LED P1.7
//
//            if (timecheck == 500) {  // if statement to determine what the state should be the next
millisecond into the Timer_A function
//                timecheck = 0;
//                P1OUT = 0;
//                statevar = 5;  // Next Timer_A call go to state 5
//            } else {
//                statevar = 4;  // stays the same.  So not really needed
//            }
//            break;
//
//        case 5:  //LED3 ON
//
//            P1OUT = 0x40;  // turn LED P1.6
//
//            if (timecheck == 500) {  // if statement to determine what the state should be the next
millisecond into the Timer_A function
//                timecheck = 0;
//                P1OUT = 0;
//                statevar = 6;  // Next Timer_A call go to state 6
//            } else {
//                statevar = 5;  // stays the same.  So not really needed
//            }
//            break;
//
//        case 6:  //LED2 ON
//
//            P1OUT = 0x20;  // turn LED P1.5
//
//            if (timecheck == 500) {  // if statement to determine what the state should be the next
millisecond into the Timer_A function
//                timecheck = 0;
//                P1OUT = 0;
//                statevar = 1;  // Next Timer_A call go to state 1
//            } else {
//                statevar = 6;  // stays the same.  So not really needed
//            }
//            break;

    if (get_switchstate() == '0') //for each case, one LED will be turned on and off to blink, and there is a
sign Blink to represent the LED state now
        {
            if (Blink == 1) //when Blink=1, turn on the LED
            {
                P1OUT = 0x10;
            }
            else
            {
                P1OUT = 0x00; //when Blink=0, turn off the LED
            }

            if (timecheck == 300)//count for 300ms, so LED will be turned on and off both for 300ms and change
atate
            {
```

```
            timecheck = 0;
            if (P1OUT == 0x10)
            {
                Blink = 0;
            }
            else
            {
                Blink = 1;
            }
        }
    }


if (get_switchstate() == '1')
{
    if (Blink == 1)
    {
        P1OUT = 0x20;
    }
    else
    {
        P1OUT = 0x00;
    }

    if (timecheck == 300)
    {
        timecheck = 0;
        if (P1OUT == 0x20)
        {
            Blink = 0;
        }
        else
        {
            Blink = 1;
        }
    }
}

if (get_switchstate() == '2')
{
    if (Blink == 1)
    {
        P1OUT = 0x40;
    }
    else
    {
        P1OUT = 0x00;
    }

    if (timecheck == 300)
    {
        timecheck = 0;
        if (P1OUT == 0x40)
        {
            Blink = 0;
        }
        else
        {
            Blink = 1;
        }
    }
}

if (get_switchstate() == '3')
{
    if (Blink == 1)
    {
        P1OUT = 0x80;
    }
    else
    {
        P1OUT = 0x00;
    }

    if (timecheck == 300)
    {
        timecheck = 0;
        if (P1OUT == 0x80)
        {
            Blink = 0;
        }
        else
        {
            Blink = 1;
            }
        }
    }

}


/*
// ADC 10 ISR - Called when a sequence of conversions (A7-A0) have completed
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void) {

}
*/


// USCI Transmit ISR - Called when TXBUF is empty (ready to accept another character)
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void) {

        if(IFG2&UCA0TXIFG) {            // USCI_A0 requested TX interrupt
                if(printf_flag) {
                        if (currentindex == txcount) {
                                senddone = 1;
                                printf_flag = 0;
                                IFG2 &= ~UCA0TXIFG;
                        } else {
                                UCA0TXBUF = printbuff[currentindex];
                                currentindex++;
                        }
                } else if(UART_flag) {
                        if(!donesending) {
                                UCA0TXBUF = txbuff[txindex];
                                if(txbuff[txindex] == 255) {
                                        donesending = 1;
                                        txindex = 0;
                                }
                                else txindex++;
                        }
                } else {  // interrupt after sendchar call so just set senddone flag since only one char is 
sent
                        senddone = 1;
                }

                IFG2 &= ~UCA0TXIFG;
        }

        if(IFG2&UCB0TXIFG) {    // USCI_B0 requested TX interrupt (UCB0TXBUF is empty)

                IFG2 &= ~UCB0TXIFG;   // clear IFG
        }
}


// USCI Receive ISR - Called when shift register has been transferred to RXBUF
// Indicates completion of TX/RX operation
#pragma vector=USCIAB0RX_VECTOR
__interrupt void USCI0RX_ISR(void) {

        if(IFG2&UCB0RXIFG) {  // USCI_B0 requested RX interrupt (UCB0RXBUF is full)

                IFG2 &= ~UCB0RXIFG;   // clear IFG
        }

        if(IFG2&UCA0RXIFG) {  // USCI_A0 requested RX interrupt (UCA0RXBUF is full)

//    Uncomment this block of code if you would like to use this COM protocol that uses 253 as STARTCHAR and 
255 as STOPCHAR
/*              if(!started) {  // Haven't started a message yet
                        if(UCA0RXBUF == 253) {
                                started = 1;
                                newmsg = 0;
                        }
                }
                else {  // In process of receiving a message
                        if((UCA0RXBUF != 255) && (msgindex < (MAX_NUM_FLOATS*5))) {
                                rxbuff[msgindex] = UCA0RXBUF;

                                msgindex++;
                        } else {        // Stop char received or too much data received
                                if(UCA0RXBUF == 255) {  // Message completed
                                        newmsg = 1;
                                        rxbuff[msgindex] = 255; // "Null"-terminate the array
                                }
                                started = 0;
                                msgindex = 0;
                        }
                }
*/

                IFG2 &= ~UCA0RXIFG;
        }

}

// This function takes care of all the timing for printing to UART
// Rate determined by how often the function is called in Timer ISR
int print_timecheck = 0;
void print_every(int rate) {
    if (rate < 15) {
        rate = 15;
    }
    if (rate > 10000) {
        rate = 10000;
    }
    print_timecheck++;
    if (print_timecheck == rate) {
        print_timecheck = 0;
        newprint = 1;
    }

}
```