```
/*******************************************************************************
 *                        MSP-EXP430G2-LaunchPad LED8 Digital Example
 *
 * Description:
 * This code provides an example for changing the digital output status. Depending
 * on which ascii character is sent to the device,  and the results also sent to the
 * computer.
 *
 * Code Modified from "A Simple ADC Example on the LaunchPad"
 *
 * 1st Author: Nicholas J. Conn - http://msp430launchpad.com
 * Email: webmaster at msp430launchpad.com
 * Date: 08-29-10
 *
 * 2nd Author: gmaxsonic - https://sites.google.com/site/msp430launchpaddiy/
 * Email: gmaxsonic at gmail.com
 * Date: 05-25-11
 *******************************************************************************/

#include       <msp430g2553.h>
#include       <stdbool.h>

#define        TXD            BIT1   // TXD on P1.1
#define        RXD            BIT2    // RXD on P1.2

#define        Bit_time       104     // 9600 Baud, SMCLK=1MHz (1MHz/9600)=104
#define        Bit_time_5     52     // Time for half a bit.

                                        // ASCII values for the commands
#define        M_D0           0x30   // Char ASCII "0"
#define        M_D3           0x33   // Char ASCII "3"
#define        M_D4           0x34   // Char ASCII "4"
#define        M_D5           0x35   // Char ASCII "5"
#define        M_D6           0x36   // Char ASCII "6"
#define        M_D7           0x37   // Char ASCII "7"


unsigned char   BitCnt;                // Bit count, used when transmitting byte
unsigned int    TXByte;                // Value sent over UART when Transmit() is called
unsigned int    RXByte;                // Value recieved once hasRecieved is set

unsigned int    i;                     // 'for' loop variable

bool           isReceiving;             // Status for when the device is receiving
bool            hasReceived;           // Lets the program know when a byte is received

/*******************************************************************************
*Function Definitions
*******************************************************************************/

void Transmit(void);
void Receive(void);
void main(void)
{
       WDTCTL = WDTPW + WDTHOLD;            // Stop WDT

       BCSCTL1 = CALBC1_1MHZ;                       // Set range
```

```
        DCOCTL = CALDCO_1MHZ;                // SMCLK = DCO = 1MHz

        P1SEL |= TXD;                        // Connected TXD to timer pin
        P1DIR |= TXD;

        P1IES |= RXD;                        // RXD Hi/lo edge interrupt
        P1IFG &= ~RXD; // Clear RXD (flag) before enabling interrupt
        P1IE |= RXD;                         // Enable RXD interrupt
        P1DIR |= BIT0+BIT3+BIT4+BIT5+BIT6+BIT7;
        P1OUT &= ~BIT0+BIT3+BIT4+BIT5+BIT6+BIT7; // Turn off LED

        isReceiving = false;                   // Set initial values
        hasReceived = false;
        __bis_SR_register(GIE);                // interrupts enabled

        while(1)
        {
                if (hasReceived)             // If the device has recieved a value
                {
                        Receive();
                        TXByte = RXByte;
                        Transmit();
                }
                if (~(hasReceived))          // Loop again if either flag is set
                         __bis_SR_register(CPUOFF + GIE);
                  // LPM0, the ADC interrupt will wake the processor up.
        }
}

/*******************************************************************************************
* Handles the received byte and calls the needed functions
*******************************************************************************************/
void Receive()
{
        hasReceived = false;                 // Clear the flag
        switch(RXByte)               // Switch depending on command value received
        {
        case M_D0:
                 P1OUT ^= BIT0;              // Toggle LED on P1.0
                 break;
      case M_D3:
                 P1OUT ^= BIT3;              // Toggle LED on P1.3
                 break;
        case M_D4:
                 P1OUT ^= BIT4;              // Toggle LED on P1.4
                 break;
      case M_D5:
                 P1OUT ^= BIT5;              // Toggle LED on P1.5
                 break;
        case M_D6:
                   P1OUT ^= BIT6;              // Toggle LED on P1.6
                 break;
        case M_D7:
                   P1OUT ^= BIT7;              // Toggle LED on P1.7
                 break;

        default:;
```

```
        }
}

/*****************************************************************************
* Transmits the value currently in TXByte. The function waits till it is
*   finished transmiting before it returns.
*****************************************************************************/
void Transmit()
{
             while(isReceiving);                 // Wait for RX completion
               CCTL0 = OUT;                      // TXD Idle as Mark
               TACTL = TASSEL_2 + MC_2;          // SMCLK, continuous mode

               BitCnt = 0xA;                      // Load Bit counter, 8 bits + ST/SP
               CCR0 = TAR;                        // Initialize compare register

               CCR0 += Bit_time;                  // Set time till first bit
               TXByte |= 0x100;                  // Add stop bit to TXByte (which is logical
1)
               TXByte = TXByte << 1;             // Add start bit (which is logical 0)

               CCTL0 = CCIS0 + OUTMOD0 + CCIE;    // Set signal, intial value, enable
interrupts
               while ( CCTL0 & CCIE );           // Wait for previous TX completion
}


/*****************************************************************************
* Port 1 interrupt service routine. Starts the receive timer, and disables any
* current transmission.
*****************************************************************************/
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
            isReceiving = true;

            P1IE &= ~RXD;                       // Disable RXD interrupt
            P1IFG &= ~RXD;                       // Clear RXD IFG (interrupt flag)

            TACTL = TASSEL_2 + MC_2;             // SMCLK, continuous mode
            CCR0 = TAR;                         // Initialize compare register
            CCR0 += Bit_time_5;                 // Set time till first bit
            CCTL0 = OUTMOD1 + CCIE;             // Dissable TX and enable interrupts

            RXByte = 0;                         // Initialize RXByte
            BitCnt = 0x9;                       // Load Bit counter, 8 bits + ST
}

/*****************************************************************************
*Timer A0 interrupt service routine.  This handles transmitting and receiving bytes.
*****************************************************************************/
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
            if(!isReceiving)
            {
                CCR0 += Bit_time;                  // Add Offset to CCR0
```

```
    if ( BitCnt == 0)                    // If all bits TXed
    {
            TACTL = TASSEL_2;    // SMCLK, timer off (for power consumption)
           CCTL0 &= ~ CCIE ;            // Disable interrupt
    }
    else
    {
            CCTL0 |= OUTMOD2;            // Set TX bit to 0
            if ( TXByte & 0x01)
            CCTL0 &= ~ OUTMOD2;          // If it should be 1, set it to 1
           TXByte = TXByte >> 1;
            BitCnt --;
    }
}
else
{
    CCR0 += Bit_time;                    // Add Offset to CCR0
    if ( BitCnt == 0)
    {
            TACTL = TASSEL_2;     // SMCLK, timer off (for power consumption)
           CCTL0 &= ~ CCIE ;           // Disable interrupt

            isReceiving = false;

            P1IFG &= ~RXD;               // clear RXD IFG (interrupt flag)
            P1IE |= RXD;                 // enabled RXD interrupt

            if ( (RXByte & 0x201) == 0x200)
                                  //Validate the start&stop bits are correct
            {
                  RXByte = RXByte >> 1; // Remove start bit
                  RXByte &= 0xFF;       // Remove stop bit
                  hasReceived = true;
             }
             __bic_SR_register_on_exit(CPUOFF);
                              // Enable CPU so the main while loop continues
      }
      else
      {
               if ( (P1IN & RXD) == RXD)// If bit is set?
               RXByte |= 0x400;          // Set the value in the RXByte
               RXByte = RXByte >> 1;     // Shift the bits down
              BitCnt --;
       }
  }
}
```