```
/*******************************************************************************
 *              LabVIEW Interface for LaunchPad (LILP_MSP430G2553) v4
 *
 * Description:
 *            This code provides an example for using the 10 bit ADC
 *            and Digital in the MSP430G2553.
 *            Depending on which ascii character is sent to the device,
 *             and the results sent to the computer.
 *
 * Author: gmaxsonic - http://sites.google.com/site/msp430launchpaddiy/
 * Email: gmaxsonic at gmail.com
 * Date: Mar-01-2012
 *
 * Reference Source Code from "A Simple ADC Example on the LaunchPad"
 * Author: Nicholas J. Conn - http://msp430launchpad.com
 * Email: webmaster at msp430launchpad.com
 * Date: 08-29-2010
 *
 * Release Note : add temperature sampling and toggle LED P1.0
 *                modify for Scratch Booster Pack v1 hardware spec
*******************************************************************************/

#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_A4          0x34    // Char ASCII "4"
#define               M_A5          0x78    // Char ASCII "x"
#define               M_A6          0x79    // Char ASCII "y"
#define               M_A7          0x7A    // Char ASCII "z"
#define       M_TEMP          0x54    // Char ASCII "T"
#define              M_VCC          0x56    // Char ASCII "V"

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

bool             ADCDone;                // ADC Done flag
unsigned int    ADCValue;               // Measured ADC Value

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

void Transmit(void);
void Receive(void);
```

```
void Measure(unsigned int);
void Measure_REF(unsigned int, unsigned int);
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;
	P1OUT &= ~BIT0;				// Turn off LED at P1.0

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

	while(1)
	{
		if (hasReceived)			// If the device has recieved a value
		{
			Receive();
		}
		if(ADCDone)				// If the ADC is done with a measurement
		{
			ADCDone = false;				// Clear flag
			TXByte = ADCValue & 0x00FF;			// Set TXByte
			Transmit();					// Send
			TXByte = (ADCValue >> 8);			// Set TXByte to the upper 8 bits
			TXByte = TXByte & 0x00FF;
			Transmit();
		//	 P1OUT ^= BIT0;			// Toggle LED on/off P1.0

		}
		if (~(hasReceived && ADCDone))		// 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/off P1.0
```

```
                break;

        case M_A4:
                Measure(INCH_4);                    // Reads A4
                break;

        case M_A5:
                Measure(INCH_5);                    // Reads A5
                break;

        case M_A6:
                Measure(INCH_6);                    // Reads A6
                break;

        case M_A7:
                Measure(INCH_7);                    // Reads A7
                break;

        case M_TEMP:
                  Measure_REF(INCH_10,0);    // Reads the temperature sensor, ref 1.5V
               P1OUT ^= BIT0;                // Toggle LED on/off P1.0
               break;

        case M_VCC:
                Measure_REF(INCH_11, REF2_5V);      // Reads VCC once (VCC/2 internally)
                break;

        default:;
        }
}

/*****************************************************************************
* Reads ADC 'chan' once using AVCC as the reference.
*****************************************************************************/
void Measure(unsigned int chan)
{
        ADC10CTL0 &= ~ENC;                           // Disable ADC
        ADC10CTL0 = ADC10SHT_3 + ADC10ON + ADC10IE;
                                                                        // 16 clock ticks,
ADC On, enable ADC interrupt
        ADC10CTL1 = ADC10SSEL_3 + chan;             // Set 'chan', SMCLK
        ADC10CTL0 |= ENC + ADC10SC;          // Enable and start conversion
}


/*****************************************************************************
* Reads ADC 'chan' once using an internal reference, 'ref' determines if the
*   1.5V reference is used.
*****************************************************************************/
void Measure_REF(unsigned int chan, unsigned int ref)
{
        ADC10CTL0 &= ~ENC;                           // Disable ADC
        ADC10CTL0 = SREF_1 + ADC10SHT_3 + REFON + ADC10ON + ref + ADC10IE;
                                    // Use reference,
                                   // 16 clock ticks, internal reference on
                                    // ADC On, enable ADC interrupt, Internal = 'ref'
        ADC10CTL1 = ADC10SSEL_3 + chan;      // Set 'chan', SMCLK
```

```
        __delay_cycles (128);               // Delay to allow Ref to settle
        ADC10CTL0 |= ENC + ADC10SC;         // Enable and start conversion
}


/*****************************************************************************
* 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

}


/*****************************************************************************
* ADC interrupt routine. Pulls CPU out of sleep mode for the main loop.
*****************************************************************************/
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR (void)
{
        ADCValue = ADC10MEM;                                // Saves measured value.
        ADCDone = true;                                     // Sets flag for main loop.
        __bic_SR_register_on_exit(CPUOFF);// Enable CPU so the main while loop continues
}


 /*****************************************************************************
 * 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 --;
                    }
                }
}
```