B RF Tag Microcontroller Code

```
//*****************************************************************************
//  Tag 2 ADC sampling & comm protocol
//
//  Description: Input A0 is sampled for the coarse CT
//               Input A1 is sampled for the fine CT
//               Input A5 is sampled for the 1.5V virtual ground
//
//  TimerB is used as a loop timer, setting a flag every 1ms.
//
//  A while loop continues perpetually to check if TBIFG is set, which
//  indicates that 1ms has gone by since the main loop was last entered.
//
//  When this logical statement is true, the coarse CT value is loaded from ADC12MEM0
//  into the variable ADCresult_0, the fine CT value is loaded from ADC12MEM1 into the
//  variable ADCresult_1, and the 1.5V sampled value is loaded from ADC12MEM3 into the
//  variable ADCresult_5.
//
//  The deviation of the 1.5V virtual ground from a true 1.5V value, which is 2047 for
//  reference voltage of 3V, since (4095)*(Vin/Vref)=(4095)*(1/2)=2047, is then computed.
//  The calculated error is removed from the coarse and fine CT values to center them
//  to a true 1.5V DC offset.
//
//  A three bit checksum is then computed for the coarse and fine ct data.
//  The three bits from each checksum are concatentated to form a six bit header.
//
//  The coarse and fine CT data is then appended with the header bits and start bits to
//  create a packet ready for transmission.  The packet is as follows:
//  MSB                                 LSB
//  CoarseCT + FineCT + Header bits + Start bits
//
//  A loop is then used to send out the packet LSB first.
//
//  When the bit to be sent out is a zero, the original spread spectrum sequence
//  is sent out.
//  When the bit to be sent out is a one, the complement of the orginial spread
//  spectrum sequence is sent out.
//
//  The transmission occurs by sending the spreading sequence a byte at a time to
//  the SPI output buffer, UCA0TXBUF.  The SPI output is driven by ACLK.
//
//  *ADC notes:  Takes 8 ADC12CLK cycles to sample input, 12 cycles to convert
//  the sample, 1 cyles to store the result
//
//  Greg Koo
//  Propagation Group, GaTech
//  September 2009
//  Built with IAR Embedded Workbench Version: 4.20
//*****************************************************************************

#include "msp430x24x.h"


// Variables in RAM.
unsigned long ADCresult_0=0;
unsigned long ADCresult_1=0;
unsigned long ADCresult_5=0;
long data_out=0;
unsigned char i=1;
unsigned char j=0;
char start_bits=0;
unsigned int header_bits=0;
unsigned int k=0;
unsigned char CAL_DC0=0;
int offset_error=0;
unsigned int ADCresult_0_crc=0;
unsigned int ADCresult_1_crc=0;
unsigned int divisor0=0xb;
unsigned int divisor1=0xb<<1;
unsigned int divisor2=0xb<<2;
unsigned int divisor3=0xb<<3;
unsigned int divisor4=0xb<<4;
```

```
unsigned int divisor5=0xb<<5;
unsigned int divisor6=0xb<<6;
unsigned int divisor7=0xb<<7;
unsigned int divisor8=0xb<<8;
unsigned int divisor9=0xb<<9;
unsigned int divisor10=0xb<<10;
unsigned int divisor11=0xb<<11;


void init_sys(void);                        // System initialization.
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;             // Stop watchdog timer
  init_sys();                               // System Initialization


  while (1)
  {
    if(TBCTL & TBIFG){
               TBCTL=TBCTL^TBIFG;
               data_out=(ADCresult_0<<20)+(ADCresult_1<<8)+(header_bits<<2)+1;
               while(i<=32){
                 switch(data_out&0x01){
                 case 0:
                 //////////////////////////INSERT SS_code_sequence_X Here

                 UCA0TXBUF=0x36;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0xCE;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0x95;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0x15;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0x7C;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0x97;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0xFD;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0x8A;


                 //////////////////////////End of SS_code sequence
                 break;
                 case 1:
                 //////////////////////////INSERT SS_code_comp_sequence Here

                 UCA0TXBUF=0xC9;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0x31;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0x6A;
                 while(!(UCA0TXIFG & IFG2))
                 {
                 }
                 UCA0TXBUF=0xEA;
                 while(!(UCA0TXIFG & IFG2))
                 {
```

```
            }
            UCA0TXBUF=0x83;
            while(!(UCA0TXIFG & IFG2))
            {
            }
            UCA0TXBUF=0x68;
            while(!(UCA0TXIFG & IFG2))
            {
            }
            UCA0TXBUF=0x02;
            while(!(UCA0TXIFG & IFG2))
            {
            }
            UCA0TXBUF=0x74;
            ////////////////////////////////End of SS_code_comp_sequence
            break;
  }
  ++i;
  data_out=data_out>>1;
}
i=1;
  __delay_cycles(550);  //push ADC sampling to end of loop
 ADC12CTL0 ^= ADC12SC;// Start ADC sampling
 ADC12CTL0 ^= ADC12SC;
 __delay_cycles(750); // wait 75us for 3 samples and conversions

offset_error= 2047-ADC12MEM2; // calc how far off from 1.5V virtual gnd is
ADCresult_0=ADC12MEM0+offset_error; // remove error from coarse ct reading
ADCresult_1=ADC12MEM1+offset_error; // remove error from fine ct reading

ADCresult_0_crc=(int)ADCresult_0<<3;
ADCresult_1_crc=(int)ADCresult_1<<3;

if (ADCresult_0_crc & 0x4000){
  ADCresult_0_crc=ADCresult_0_crc^divisor11;

}if (ADCresult_0_crc& 0x2000){
  ADCresult_0_crc=ADCresult_0_crc^divisor10;

}if (ADCresult_0_crc& 0x1000){
  ADCresult_0_crc=ADCresult_0_crc^divisor9;

}if (ADCresult_0_crc & 0x800){
  ADCresult_0_crc=ADCresult_0_crc^divisor8;

}if (ADCresult_0_crc& 0x400){
  ADCresult_0_crc=ADCresult_0_crc^divisor7;

}if (ADCresult_0_crc& 0x200){
  ADCresult_0_crc=ADCresult_0_crc^divisor6;

}if (ADCresult_0_crc& 0x100){
  ADCresult_0_crc=ADCresult_0_crc^divisor5;
}if (ADCresult_0_crc& 0x080){
  ADCresult_0_crc=ADCresult_0_crc^divisor4;
}if (ADCresult_0_crc& 0x040){
  ADCresult_0_crc=ADCresult_0_crc^divisor3;
}if (ADCresult_0_crc& 0x020){
  ADCresult_0_crc=ADCresult_0_crc^divisor2;

}if (ADCresult_0_crc& 0x010){
  ADCresult_0_crc=ADCresult_0_crc^divisor1;

}if (ADCresult_0_crc& 0x008){
  ADCresult_0_crc=ADCresult_0_crc^divisor0;
}

if (ADCresult_1_crc & 0x4000){
  ADCresult_1_crc=ADCresult_1_crc^divisor11;

}if (ADCresult_1_crc& 0x2000){
  ADCresult_1_crc=ADCresult_1_crc^divisor10;

}if (ADCresult_1_crc& 0x1000){
  ADCresult_1_crc=ADCresult_1_crc^divisor9;

}if (ADCresult_1_crc & 0x800){
```

```
        ADCresult_1_crc=ADCresult_1_crc^divisor8;

      }if (ADCresult_1_crc& 0x400){
        ADCresult_1_crc=ADCresult_1_crc^divisor7;

      }if (ADCresult_1_crc& 0x200){
        ADCresult_1_crc=ADCresult_1_crc^divisor6;

      }if (ADCresult_1_crc& 0x100){
        ADCresult_1_crc=ADCresult_1_crc^divisor5;
      }if (ADCresult_1_crc& 0x080){
        ADCresult_1_crc=ADCresult_1_crc^divisor4;
      }if (ADCresult_1_crc& 0x040){
        ADCresult_1_crc=ADCresult_1_crc^divisor3;
      }if (ADCresult_1_crc& 0x020){
        ADCresult_1_crc=ADCresult_1_crc^divisor2;

      }if (ADCresult_1_crc& 0x010){
        ADCresult_1_crc=ADCresult_1_crc^divisor1;

      }if (ADCresult_1_crc& 0x008){
        ADCresult_1_crc=ADCresult_1_crc^divisor0;
      }

    header_bits=ADCresult_1_crc+(ADCresult_0_crc<<3);
    }

  }
}

void init_sys(void)
{
  SVSCTL=0xB0+PORON;
  while(SVSCTL&SVSFG){ //loop while brownout flags occur
  SVSCTL=SVSCTL&(!SVSFG);
  }
  CAL_DC0=CALBC1_1MHZ&0x0F;
  DCOCTL = CALDCO_1MHZ; // Set DCO to 1 MHz
  BCSCTL1=XT2OFF+XTS+CAL_DC0+DIVA_2; // Sets XT1 to HF mode, Turns off XT2,
                                     //Sets DCO to 1MHz,
                                     // ACLK = external /4 (2.5MHz)
  BCSCTL3= LFXT1S_2+XCAP_0    ; // 1pF XT1 load, XT1 set to support up to 16MHZ osc.

  IFG1 &= ~OFIFG; //clr osc. fault falg

    while (IFG1 & OFIFG){
      IFG1 &= ~OFIFG;
    }//Wait until oscillator has stabilized

  BCSCTL2 = SELM_3;// Select XT1 as MCLK (External @ 10 MHz)

  // Configure all unused port pins as outputs.
  P1DIR = 0xff;                             // Set port to outputs
  P1OUT = 0;                                // Reset port outputs
  P2DIR = 0xff;                             // Set port to outputs
  P2OUT = 0;                                // Reset port outputs
  P3DIR = 0xfe;                             // Set port to outputs
  P3OUT = 0;                                // Reset port outputs
  P3SEL |= BIT4;                            // P3.4 set as SPI Data OUT
  P4DIR = 0xff;                             // Set port to outputs
  P4OUT = 0;                                // Reset port outputs
  P5DIR = 0xff;                             // Set port to outputs
  P5OUT = 0;                                // Reset port outputs
  P6SEL = 0x03;                             // P6.0 = analog input
  P6DIR = 0xdc;                             // set unused pins to outputs
  P6OUT = 0;

  // Configure ADC
  ADC12CTL0 = ADC12ON+MSC+SHT0_1;       // Setup ADC12, ref., sampling time, multi sample & conv,
                                        // 8 ADC12CLK cyles for sample period
  ADC12CTL1 = CSTARTADD_0+ADC12SSEL_3+SHP+CONSEQ_1;      // TB1 sampling timer,ADC12CLK=SMCLK
  ADC12MCTL0 = SREF_2|INCH_0;                       // sample coarse CT
  ADC12MCTL1 = SREF_2+INCH_1;                       // sample fine CT
  ADC12MCTL2 = SREF_2+EOS+INCH_5;                       // sample 1.5V virtual ground

  // Timer B config
```

```
  TBCTL = TBSSEL_2 | MC_1;                  // TBCLK = SMCLK, Up mode.
  TBCCR0=1000;

  // Start ADC conversion
  ADC12CTL0 |= ENC;
  ADC12CTL0 ^= ADC12SC;
  ADC12CTL0 ^= ADC12SC;

  // Configure SPI
  UCA0CTL1=UCSWRST;
  UCA0CTL0=UCMST+UCSYNC+UCMSB;                          // SPI master mode
  UCA0CTL1=UCSSEL_1 ;               // intialize SPI, clk=ACLK

} // sys_init()
```