We are
Delivering M2M
around the World.

TRC103

FHSS

By

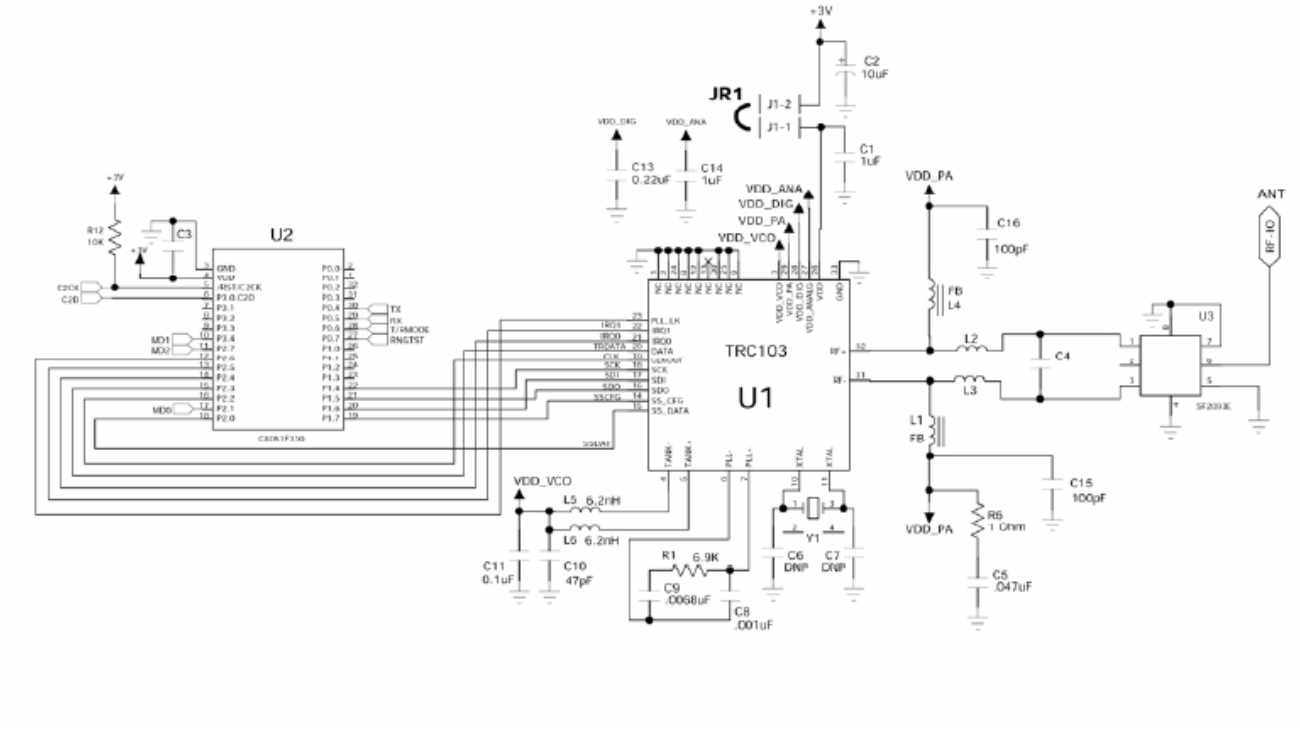
Bob Nelson

8/15/2008

Scope:

This application will demonstrate the ease of implementing a FHSS radio when using the TRC103. This document will give you the schematic and firmware to implement the design. Keep in mind that the TRC103 will meet or exceed FCC 15.249 power requirements and this application note is intended to use the higher power out while the radio is being used has a FHSS mode meeting FCC 15.247 rules and regulations.

Schematic:



Theory:

According to FCC 15.247 rules, the transmitter can not dwell on any giver channel longer than 400ms. Also the FCC requires you to hop to at lease 25 channels during your operation. The firmware below uses the TRC103 RFIC to accomplish this with ease do to the fast channel switching time within the transceiver. The firmware is sending a short range test packet which you can modify to add your data. During operation the channel that the transmitter is going to switch to will follow the packet number being sent within the packet protocol to make the synchronization simple and fast. The micro that is used in this demo is the Silabs 330F utilizing the internal clock oscillator.

Firmware:

Text in red is FHSS code.

```
#include <c8051f310.h>        /* Include register definition file.*/
#include <string.h>           /* Include string functions file.*/
#include <stdlib.h>           /* Include string functions file.*/
#include <intrins.h>
#include <stdio.h>            /* Include string functions file.*/
#include <F310_FlashPrimitives.h>
#include <F310_FlashUtils.h>

//-----------------------------------------------------------------------------
//                      Global CONSTANTS
//-----------------------------------------------------------------------------
#define             Start     1
#define             Stop      0
#define             Yes       1
#define             No        0
#define             STX       02h
#define             ETX       03h
#define   XON       0x11
#define   XOFF      0x13


typedef unsigned char uchar;
typedef unsigned int uint;

//-----------------------------------------------------------
//          PORT ASSIGNMENTS
//-----------------------------------------------------------

sbit        PLLCK   =   P2^6;           // PLL LOCK IND
sbit        IRQ0    =   P2^4;           // INTERRUPT
sbit        IRQ1    =   P2^5;           // INTERRUPT
sbit        DAT     =   P2^3;           // DATA TO/FROM RF
sbit        CLK     =   P2^2;           // CLKOUT
sbit        SSDAT   =   P2^0;           // DATA SELECT
sbit        SSCFG   =   P1^7;           // SPI CONFIG SELECT
sbit        SDI     =   P1^6;           // SPI SERIAL IN TO RF
sbit        SDO     =   P1^5;           // SPI SERIAL OUT FROM RF
sbit        SCK     =   P1^4;           // SPI SERIAL CLOCK TO RF
//sbit      STX     =   P0^4;           // USART TX
//sbit      SRX     =   P0^5;           // USART RX
sbit        PB1     =   P0^6;           // MODE SELECT PB (TX,RX,SLEEP) /INT1
sbit        PB2     =   P0^7;           // RANGE TEST SELECT /INT0
```

```
sbit        TXPWR  =   P3^1;              // ANALOG INPUT
sbit        ErrLED   =   P0^0;             // D3
sbit        CmdLED=   P0^1;              // D2
sbit        ComLED=   P0^2;              // D1
sbit        GoodLED=   P0^3;              // D4
sbit        TXMD    =   P2^7;             // LED TX MODE SELECT
sbit        RXMD    =   P3^4;             // LED RX MODE SELECT
sbit        RTMD    =   P2^1;             // LED RANGE TEST MODE SELECT

//                                      =   P1^0;              // not used
//                                      =   P1^1;              // not used
//                                      =   P1^2;              // not used
//                                      =   P1^3;              // not used
//                                      =   P3^2;              // not used
//                                      =   P3^3;              // not used

/*
//-----------------------------------------------------------------------------
//                          Function PROTOTYPES
//-----------------------------------------------------------------------------
void PCA_Init(void);
void Reset_Sources_Init(void);
void Ports_Init(void);
void Oscillator_Init(void);
void Interrupts_Init(void);
void SPI_Init(void);
void HWU_INIT(void);
void Timer_Init(void);
void ADC_Init(void);
void Vref_Init(void);
void UART_ISR (void);
void RANGETST_ISR (void);
void MODECHG_ISR (void);*/

void Hop(void);
void LED_Init (void);
void Config_Start (void);
void Config_TX (void);
void Config_RX (void);
void Config_SLP(void);
void SPI_CFG (uchar addr,uchar dat);
void mSecDly (uint mSecDlyCnt);
void Range_TX (void);
void Range_RX (void);
//void Verify (void);
void Range_Config (void);
void Tmr0 (uchar rTH0,uchar rTL0);
void Ack(void);
void nAck(void);
void Read(void);
uchar ASC2HEX (uchar byte);
void HEX2ASC (uchar byte,uchar *pByte1,uchar *pByte2);
void ReadCfg(void);
void FLASH_ByteWrite (FLADDR addr, char byte);
//void FLASH_Clear (FLADDR addr, unsigned numbytes);
uchar FLASH_ByteRead (FLADDR addr);
void FLASH_PageErase (FLADDR addr);
void Do_TX(uchar *pMem, uchar len);
void Wait_for_SPIF(void);
void Wait_for_UART_Tx(void);
void Wait_for_UART_Rx(void);
uchar Read_Reg(uchar addr);
```

```
void SetforRx(void);
void SetforTx(void);
void Rst_CRC(void);
void Clear_FIFO(void);


//-----------------------------------------------------------------------------
//                      Global Variables
//-----------------------------------------------------------------------------
uchar     MODE;                                                 // Set to: 0=RX
                                                                //         1=TX
                                                                //         2=SLEEP

uchar     cfgAddr;                                          // Address Reg for read/write Config info
                                                                //    START(0) | R/nW | A4 | A3 | A2 |
A1 | A0 | STOP(0)
                                                                // READ = 0x40-0x7E
                                                                // WRITE = 0x00-0x3E

uchar     t3cnt;                                // ADC conversion timer
uchar     cfgcnt;                               // counter for reading config data from RFDA
uchar     bdata     status;                     // Bit addressable status register
          sbit fUpdate = status^0;              // Update configuration parameters Flag
          sbit fRTESTx = status^1;              // Range test initiator on=1/off=0 Flag
          sbit fRTESTRx = status^2;             // Range test receive on=1/off=0 Flag
          sbit fTermAct = status^3;             // Terminal active flag
          sbit fFlshClr = status^4;             // Clear flash flag
          sbit fTermDat = status^5;             // Terminal Data active flag
          sbit fWaitforAck = status^6;  // Wait for Ack/nAck flag - this flag keeps from Ack'ing and Ack

sfr16 ADCval = 0xBD;                            // ADCval = ADC0H(0xBD) and ADC0L(0xBE)

uchar *pBuf;                                    // 8-bit pointer to access buffer memory
uchar xdata Buffer[129];                        // Allocate Buffer memory
uchar xdata Reg_Buf[64];                        // Buffer to hold Reg settings

static const uchar xdata rangeTestStr[] = "\x02RF Monolithics TRC103 Range Test Demo.\x03";    //  String
static const uchar xdata AckStr[]  = "ACK";             // Acknowledge String
static const uchar xdata nAckStr[] = "NACK";            // Error String
uchar code Regs[62] _at_ 0x2000;                        // Memory area for storing TRC103 Register values

uchar     hop_channel;
uchar     current_reg;
unsigned int hhtime ;                                   // 40 = 357ms
unsigned int mhtime ;                                   // 19 = 169ms

static const uchar code freq_hop[25][3] = {                     // Hoop table for 25 channels
          0x76,0x62,0x37,               // 905.142 Mhz
          0x76,0x63,0x11,     // 909.620
          0x8F,0x77,0x39,               // 905.700
          0x8f,0x77,0x47,               // 907.100
          0x8f,0x78,0x26,               // 911.300
          0x8f,0x78,0x0a,               // 908.500
          0x93,0x7b,0x45,               // 911.578
          0x93,0x7b,0x2e,               // 909.340
          0x58,0x49,0x2e,               // 905.420
          0x79,0x65,0x42,               // 910.740
          0x87,0x71,0x19,               // 907.941
          0x87,0x71,0x3e,               // 911.858
          0x71,0x5e,0x36,               // 906.821
          0x51,0x44,0x00,               // 908.780
          0x75,0x61,0x4a,               // 905.979
```

```
            0x73,0x60,0x39,             // 910.179
            0x9f,0x85,0x20,             // 907.380
            0x7f,0x6a,0x3f,             // 909.900
            0x82,0x6c,0x48,             // 906.540
            0x61,0x51,0x32,             // 911.020
            0x5d,0x4e,0x00,             // 907.659
            0x92,0x7a,0x37,             // 909.061
            0x89,0x72,0x3c,             // 906.260
            0x69,0x58,0x1B,             // 910/460
            0x7e,0x69,0x3c};    // 908.220 Mhz


//-----------------------------------------------------------------------
//                      Main Function
//-----------------------------------------------------------------------
//                      Configuration
//-----------------------------------------------------------------------
void main(void)
{
            uint  ADCvalLast;                   // Value of previous ADC conversion value
            uchar datCntr = 0x00;               // data counter for terminal msgs
            uchar tmp_buf;
            uint  i;                            // generic cntr
            pBuf = Reg_Buf;                     // assign ptr addr
            cfgcnt = 0;                         // initialize cfg data count


//  PCA_Init();                                 // Disable Watchdog
    PCA0MD    &= ~0x40;
    PCA0MD    = 0x00;

//  Reset_Sources_Init();                       // Config Reset/Power-up
    VDM0CN    = 0x80;                           // VDDMON enabled
    for (i=0x00AF; i>0 ; i--);                  // Wait 100us for initialization (571nsec/tick)
    RSTSRC    = 0x02;                                   // Sets VDDMON as a reset source

//  Ports_Init();                                       // Config Ports
    P3MDIN    = 0xFD;                                   // PORT3 INPUTS-0,1,2,3
    P0MDOUT   = 0x1F;                                   // PORT0 OUTPUTS-0,1,2,3,4
    P1MDOUT   = 0xD0;                                   // PORT1 OUTPUTS-7,6,4
    P2MDOUT   = 0x83;                                   // PORT2 OUTPUTS-0,1,7
    P3MDOUT   = 0x10;                                   // PORT3 OUTPUTS-4
    P0SKIP    = 0xCF;                                   // PORT0 SKIPS
    P1SKIP    = 0x0F;                                   // PORT1 SKIPS
            P0                  = 0xF0;                 // 1111 0000
            P1                  = 0xFF;                 // 1111 1111
            P2                  = 0x7D;                 // 0111 1101
            P3                  = 0xEF;                 // 1110 1111
    XBR0      = 0x03;                                   // UART & SPI ENABLED
    XBR1      = 0xC0;                                   // PULLUPS DIS (7),XBAR ENABLED (6)

//  Interrupts_Init();                                  // Config Interrupt
            EIE1      = 0x80;
    IT01CF    = 0x76;                                   // Ext Int's 'Active Low' on port P0.6,P0.7
    IE        = 0x05;                                   // Enable Interrupts

//  Oscillator_Init();                                  // Config System clock
    OSCICN    = 0x83;                                   // SYSCLK = 24.5 MHz

//  SPI_Init();                                         // Config SPI
    SPI0CFG   = 0x40;
```

```
   SPI0CN    = 0x01;                                   // Set SPI to 3-wire Master w/ nSS disabled
   SPI0CKR   = 0x07;                                   // SPI clock=1.53 MHz (0.65 usec SCLK)

//  Timer_Init();                                      // Config Timers
   TCON      = 0x05;
   TMOD      = 0x21;
//    TMR3CN    = 0x04;                                // Enable Timer 3, 16-bit auto re-load

//  Vref_Init();                                       // Config Voltage Ref for ADC
   REF0CN    = 0x0C;

//  ADC_Init();                                        // Config ADC
   AMX0P     = 0x11;                                  // Connect to Potentiometer
//    AMX0P     = 0x1E;                                // Connects ADC to internal Temp Sensor
   AMX0N     = 0x1F;
   ADC0CN    = 0xC0;

//  HWU_INIT();                                        // Config UART(Do last b/c Timer1)
          PCON |= 0x80;                               // SMOD=1 (HW_UART uses Timer 1 overflow
                                                      // with no divide down).
          TMOD = 0x20;                                // Configure Timer 1 for use by HW_UART
          CKCON |= 0x08;                              // Timer 1 derived from SYSCLK(24.5 MHz)
          TH1 = 0x96;                                 // Timer 1 reload value (115.2kbps)
          TL1 = 0x00;                                 // Timer 1 initial value

          TR1 = Start;                                // Start Timer 1

          RI0=0;                                      // Clear HW_UART RX interrupt flag
          TI0=0;                                      // Clear HW_UART TX done interrupt flag

          SCON0 = 0x50;                         // Configure HW_UART for mode 1, Receiver enabled.

          EA = 1;                                     // Enable All Active Interrupts
          ES0 = 1;                                    // Enable UART Int

          LED_Init();                                 // Sequence/Flash LEDs on startup
          Config_RX();                                // Configure device for RX mode on startup
//-----------------------------------------------------------------------------
//                    Main Loop
//-----------------------------------------------------------------------------

          TMR3CN = 0x04;                              // Start timer for master tick
          EA = 1;

          while(1)                                    // Loop Here
          {
                    EA = 1;

                    if(fRTESTx)                                 // Test if Range test ON?
                    {
                              Range_Config();                   // Do Configuration once

                              while(fRTESTx)                    // Test if Range test still ON?

                              {
                               if (hhtime >= 40)
                                 {
                                          hhtime = 0;
                                          Hop();
                                         }

                              Range_TX();                 // Set for TX and send Range Test Packet
```

```
                                Range_RX();                     // Set for RX to receive ACK or NACK
                                Tmr0(255,255);                  // Start timer (128msec) to bail out if no data

                                while(IRQ0 == 0 && fRTESTx && TF0 == 0)   // If DataNOTRdy and
timerNOTdone, loop?
                                {
                                        // do nothing
                                }

                                TR0 = 0;                                  // Turn off Timer
                                mSecDly(35);

                                if(IRQ1 == 1)                             // CRC pass
                                {
                                        GoodLED = 1;
                                        TF0 = 0;                          // reset flag
                                        Read();
                                        Rst_CRC();
                                        Clear_FIFO();
                                        GoodLED = 0;
                                        Hop();
                                        hhtime = 0;
                                }

                                else if(IRQ1 == 0 || TF0 == 1)            // CRC failed or Timed out
                                {
                                        TF0 = 0;                                  // reset flag
                                        ErrLED = 1;
        // Turn on LED
                                        mSecDly(5);                               // Delay
                                        ErrLED = 0;
        // Turn off LED
                                        Clear_FIFO();
                                        Hop();
                                }

                                tmp_buf = Read_Reg(0x0E); // Read contents of register in TRC103
                                tmp_buf = (tmp_buf | 0x40); // Reset Pattern Recognition
                                SPI_CFG(0x0E,tmp_buf);                    // write SPI
                        //      SPI_CFG(0x0E,0x51);                       // Reset Pattern Recognition

                        }
                }

                if(fRTESTRx)                                      // Test if Range test ON?
                {
                        Range_Config();                           // Do Configuration once
                        Range_RX();                               // Set for RX

                        while(fRTESTRx)                           // Test if Range test still ON?

                        {
                         if (mhtime >= 5)
                           {
                                mhtime = 0;
                                Hop();
                                }
                                if(IRQ0==1 && fRTESTRx==1)                // If datrdy, loop?
                                {
                                        if(IRQ1==1 && fRTESTRx==1)        // CRC pass
                                        {
                                                GoodLED = 1;
```

```
                                        Read();
                                        Ack();              // Send ACK Packet and flash LED(s)
                                        GoodLED = 0;
                                        Rst_CRC();
                                        Clear_FIFO();
                                        Hop();
                                        mhtime = 0;
                                }

                                else if(IRQ1==0 && fRTESTRx==1)   // CRC failed or Timed out
                                {
                                        ErrLED = 1;
                                        nAck();             // Send nACK Packet and flash LED(s)
                                        ErrLED = 0;
                                        Rst_CRC();
                                        Clear_FIFO();
                                        Hop();
                                }
                                Range_RX();                         // Set for RX
                                tmp_buf = Read_Reg(0x0E); // Read contents of register in TRC103
                                tmp_buf = (tmp_buf | 0x40); // Reset Pattern Recognition
                                SPI_CFG(0x0E,tmp_buf);              // write SPI
                        //      SPI_CFG(0x0E,0x51);                 // Reset Pattern Recognition
                                }
                }
}

if(fTermDat)                                            // Stay in active terminal until done
{
        //Range_Config();                               // Set up for Rx Packet mode
        SetforRx();                                     // Set for RX mode
        SPI_CFG(0x0D,0x08);                             // IRQ setups

        while(fTermDat)                                 // Stay in active terminal until done
        {
                if(IRQ0)                                // Look for data rdy..
                {
                        if(IRQ1==1 && fTermDat==1)                  // CRC pass
                        {
                                //if(fWaitforAck == 0)
                                //{
                                        GoodLED = 1;
                                        Read();
                                        //Ack();
                                        Rst_CRC();
                                        Clear_FIFO();
                                        GoodLED = 0;
                        //              }
                                        /*else
                                        {
                                                GoodLED = 1;
                                                mSecDly(50);
                                                GoodLED = 0;
                                                fWaitforAck = 0;
                                        }*/
                        }

                        else                    // CRC failed or Timed out
                        {
                                //if(fWaitforAck == 0)
                                //{
                                        ErrLED = 1;
```

```
                                        nAck();
                                        Rst_CRC();
                                        Clear_FIFO();
                                        ErrLED = 0;
                                //      }
                                        /*else
                                        {
                                                ErrLED = 1;
                                                mSecDly(50);
                                                ErrLED = 0;
                                                fWaitforAck = 0;
                                        }*/

                                }

                                tmp_buf = Read_Reg(0x0E); // Read contents of register in TRC103
                                tmp_buf = (tmp_buf | 0x40); // Reset Pattern Recognition
                                SPI_CFG(0x0E,tmp_buf);          // write SPI
                        //      SPI_CFG(0x0E,0x51);             // Reset Pattern Recognition
                                SetforRx();                     // Set for RX mode
                                SPI_CFG(0x0D,0x08);             // IRQ setups
                                MODE=0;
        // RX mode configure
                                TXMD=0;
        // Turn off TX LED
                                RXMD=1;
        // Turn on RX LED

                        }
                }
        }


/*****************************************************************************/
/*              Update registers                                             */
/*****************************************************************************/

        if(fUpdate)
        {
                uchar tmp1,tmp2;
                uchar datCntr = 0x1E;
                uchar *pReg;                    // variable ptr for accessing code memory

                pBuf = Reg_Buf;

                while(datCntr)                  // Write buffered config to SPI
                {
                        tmp1 = *pBuf;
                        pBuf++;

                        tmp2 = *pBuf;
                        pBuf++;

                        SPI_CFG(tmp1,tmp2);
                        datCntr--;
                }

                pBuf = Reg_Buf;
                pReg = Regs;
                FLASH_Clear (0x2000,70);                // Prefill Flash to write new info
                FLASH_Write (Regs, pBuf, 62);
                pBuf = Reg_Buf;
```

```
                        fUpdate=No;
                }

/*****************************************************************************/
/*              Compare A/D conversion of potentiometer for TX power level   */
/*****************************************************************************/

                if(AD0INT)                                              // if conversion done
                {
                        if(ADCvalLast != (ADCval & 0xFFFC))
                        {
                                if(ADCval < 0x00AA)
                                {
                                        SPI_CFG(0x1A,0x7A);             // Set TX pwr to lowest
                                }
                                else if(ADCval < 0x0154)
                                {
                                        SPI_CFG(0x1A,0x78);             // Set TX pwr
                                }
                                else if(ADCval < 0x01FE)
                                {
                                        SPI_CFG(0x1A,0x76);             // Set TX pwr
                                }
                                else if(ADCval < 0x02A8)
                                {
                                        SPI_CFG(0x1A,0x74);             // Set TX pwr
                                }
                                else if(ADCval < 0x0352)
                                {
                                        SPI_CFG(0x1A,0x72);             // Set TX pwr
                                }
                                else if(ADCval <= 0x03FF)
                                {
                                        SPI_CFG(0x1A,0x70);             // Set TX pwr to HIGHEST
                                }
                                ADCvalLast = ADCval & 0xFFFC;
                        }

                }

/*****************************************************************************/
/*              Do A/D conversion of potentiometer 4x/sec (250 msec)         */
/*****************************************************************************/

                if(t3cnt > 28)                                          // if conversion done
                {
                        t3cnt = 0x00;                                   // Reset count
                        AD0BUSY = 1;                                    // Initiate ADC conversion
                }

        }//End while

} //End main


/*****************************************************************************************************/
/***********************************     SUBROUTINES     *********************************************/
/*****************************************************************************************************/

//-----------------------------------------------------------------------------
//      LED_Init: Sequence and Flash LEDs on startup/powerup
//-----------------------------------------------------------------------------
```

```
void LED_Init(void)
{
		ComLED=1;				// 1 LED on
	mSecDly(50);
		CmdLED=1;
	mSecDly(50);
		ErrLED=1;
	mSecDly(50);
		GoodLED=1;

	mSecDly(50);

		ComLED=1;				// 1 LED on
		CmdLED=1;
		ErrLED=0;
		GoodLED=0;

	mSecDly(50);

		ComLED=1;				// 2 LEDs on
		CmdLED=1;
		ErrLED=0;
		GoodLED=0;
		TXMD=1;


	mSecDly(50);

		ComLED=1;				// 3 LEDs on
		CmdLED=1;
		ErrLED=1;
		GoodLED=0;
		RXMD=1;
		TXMD=0;

	mSecDly(50);

		ComLED=1;				// 4 LEDs on
		CmdLED=1;
		ErrLED=1;
		GoodLED=1;
		RTMD=1;


	mSecDly(50);
		ComLED=0;				// All LEDs off
		CmdLED=0;
		ErrLED=0;
		GoodLED=0;
		RXMD=0;
		RTMD=0;
		TXMD=0;

	mSecDly(50);

		ComLED=1;				// 4 LEDs on
		CmdLED=1;
		ErrLED=1;
		GoodLED=1;
		RXMD=1;
		RTMD=1;
		TXMD=0;
```

```
        mSecDly(50);
                ComLED=0;                                       // All LEDs off
                CmdLED=0;
                ErrLED=0;
                GoodLED=0;
                RXMD=0;
                RTMD=0;
                TXMD=0;

        mSecDly(50);
                ComLED=1;                                       // 4 LEDs on
                CmdLED=1;
                ErrLED=1;
                GoodLED=1;
                RXMD=0;
                RTMD=1;
                TXMD=1;

        mSecDly(50);
                ComLED=0;                                       // All LEDs off
                CmdLED=0;
                ErrLED=0;
                GoodLED=0;
                RXMD=0;
                RTMD=0;
                TXMD=0;

        mSecDly(50);
                ComLED=1;                                       // 4 LEDs on
                CmdLED=1;
                ErrLED=1;
                GoodLED=1;
                RXMD=1;
                RTMD=1;
                TXMD=1;

        mSecDly(50);
                ComLED=0;                                       // All LEDs off
                CmdLED=0;
                ErrLED=0;
                GoodLED=0;
                RXMD=0;
                RTMD=0;
                TXMD=0;
        mSecDly(300);

}

//-------------------------------------------------------------------------------------
// Config_TX: Configure device for TX mode.
// Address Reg for writing Config info:
//
//                                  START(0) | R/nW | A4 | A3 | A2 | A1 | A0 | STOP(0)
//
//          READ = 0x40-0x7E
//          WRITE = 0x00-0x3E
//-------------------------------------------------------------------------------------
void Config_TX (void)
{
        MODE=1;                                                 // TX mode configure
        TXMD=1;                                                 // Turn on TX LED
```

```
	RXMD=0;                                                 // Turn off RX LED
	SPI_CFG(0x00,0x00);                                     // Set for sleep
	SPI_CFG(0x00,0x88);                                     // Set for TX,915-928 band,freq 1
	SPI_CFG(0x01,0x88);                             // Set to FSK,Continuous Mode,dIS PKT handling
	SPI_CFG(0x02,0x07);                                     // Set Freq dev = 50kHz
	SPI_CFG(0x03,0x07);                                     // Data Rate = 25000bps
//	SPI_CFG(0x04,0x0C);                                     // Default
	SPI_CFG(0x05,0x1F);                                     // Set FIFO depth to 64 bytes
	SPI_CFG(0x06,0x5F);                                     // Set freq1 to 916.5 MHz
	SPI_CFG(0x07,0x50);                                     //                      "
	SPI_CFG(0x08,0x23);                                     //                      "
	SPI_CFG(0x09,0x6B);                                     // Default
	SPI_CFG(0x0A,0x59);                                     // Default
	SPI_CFG(0x0B,0x3C);                                     // Default
	SPI_CFG(0x0C,0x00);                                     // Set PA rise/fall to 3usec.
	SPI_CFG(0x0D,0x00);                                     // Set IRQ0=Data,IRQ1=DCLK
	SPI_CFG(0x0E,0x11);                                     // PLL lock on pin 23
//	SPI_CFG(0x0F,0x00);                                     // Default
	SPI_CFG(0x10,0xA3);                                     // Set BWFilt
	SPI_CFG(0x11,0x38);                                     // Set PolyFilt
	SPI_CFG(0x12,0x38);                             // Set DCLK ena,32-bit pattern,Pattern Recog ena
	SPI_CFG(0x13,0x07);                                     // Default
//	SPI_CFG(0x14,0x00);                                     // Default
//	SPI_CFG(0x15,0x00);                                     // Default
	SPI_CFG(0x16,0xE2);                                     // Set Synch pattern3
	SPI_CFG(0x17,0xE2);                                     // Set Synch pattern2
	SPI_CFG(0x18,0xE2);                                     // Set Synch pattern1
	SPI_CFG(0x19,0xE2);                                     // Set Synch pattern0
	SPI_CFG(0x1A,0x70);                                     // Set TxInterpFilt
	SPI_CFG(0x1B,0xBC);                                     // ClkOut DIS, 1.6MHz
	SPI_CFG(0x1C,0x00);                                     // Set Pkt length to 64bytes
//	SPI_CFG(0x1D,0x00);                                     // Default
	SPI_CFG(0x1E,0x48);                             // 4 byte Preamb,Whiten OFF, CRC OFF
	SPI_CFG(0x1F,0x00);                                     // Do not clear FIFO if CRC fails

	if(!PLLCK)
	{
		ComLED = 1;
		while(PLLCK == 0)
		{
		}
	}
	ComLED = 1;
	mSecDly(50);
	ComLED = 0;


}

//-------------------------------------------------------------------------------------
// Config_RX: Configure device for RX continuous mode.
// Address Reg for writing Config info:
//
//                                  START(0) | R/nW | A4 | A3 | A2 | A1 | A0 | STOP(0)
//
//	READ = 0x40-0x7E
//	WRITE = 0x00-0x3E
//-------------------------------------------------------------------------------------
void Config_RX (void)
{
	MODE = 0;                                               // RX mode
	RXMD = 1;                                               // LED On
```

```
        TXMD = 0;                                       // LED Off
        SPI_CFG(0x00,0x00);                             // Set for sleep
        SPI_CFG(0x00,0x68);                             // Set for RX,915-928 band,freq 1
        SPI_CFG(0x01,0x88);                 // Set to FSK,Continuous Mode,dIS PKT handling
        SPI_CFG(0x02,0x07);                             // Set Freq dev = 50kHz
        SPI_CFG(0x03,0x07);                             // Data Rate = 2000bps
//      SPI_CFG(0x04,0x0C);                             // Default
        SPI_CFG(0x05,0x1F);                             // Set FIFO depth to 64 bytes
        SPI_CFG(0x06,0x5F);                             // Set freq1 to 916.5 MHz
        SPI_CFG(0x07,0x50);                             //                  "
        SPI_CFG(0x08,0x23);                             //                  "
        SPI_CFG(0x09,0x6B);                             // Default
        SPI_CFG(0x0A,0x59);                             // Default
        SPI_CFG(0x0B,0x3C);                             // Default
        SPI_CFG(0x0C,0x00);                 // Set PA rise/fall to 3usec,lower RX current
        SPI_CFG(0x0D,0x00);                             // Set IRQ0=Data,IRQ1=DCLK
        SPI_CFG(0x0E,0x11);                             // PLL lock on pin 23
//      SPI_CFG(0x0F,0x00);                             // Default
        SPI_CFG(0x10,0xA3);                             // Set BWFilt
        SPI_CFG(0x11,0x38);                             // Set PolyFilt
        SPI_CFG(0x12,0x38);                 // Set DCLK ena,32-bit pattern,Pattern Recog ena
        SPI_CFG(0x13,0x07);                             // Default
//      SPI_CFG(0x14,0x00);                             // Default
//      SPI_CFG(0x15,0x00);                             // Default
        SPI_CFG(0x16,0xE2);                             // Set Synch pattern3
        SPI_CFG(0x17,0xE2);                             // Set Synch pattern2
        SPI_CFG(0x18,0xE2);                             // Set Synch pattern1
        SPI_CFG(0x19,0xE2);                             // Set Synch pattern0
        SPI_CFG(0x1A,0x72);                             // Set TxInterpFilt,TX pwr max
        SPI_CFG(0x1B,0xBC);                             // ClkOut DIS, 1.6MHz
        SPI_CFG(0x1C,0x00);                             // Set Pkt length to 64bytes
//      SPI_CFG(0x1D,0x00);                             // Default
        SPI_CFG(0x1E,0x68);                 // Fx lngth mode,4 byte Preamb,Whiten OFF, CRC OFF
        SPI_CFG(0x1F,0x00);                             // Do not clear FIFO if CRC fails


//*****  For Ultra Low Power Mode  ************
//      SPI_CFG(0x1F,0x81);                             // Select page 1
//      SPI_CFG(0x15,0x81);                             // Reduce RX current 2.5mA
//      SPI_CFG(0x1F,0x80);                             // Select page 0
//***********************************************


        if(!PLLCK)
        {
                ComLED = 1;
                while(PLLCK == 0)
                {
                }
        }
        ComLED = 1;
        mSecDly(50);
        ComLED = 0;

}

//---------------------------------------------------------------------------------------
// Range_Config: Configure device for Range Test mode.
// This routine will configure the device for Buffered mode.
// After it is sent, the device will configure to RX mode to receive
// the acknowledge.
//---------------------------------------------------------------------------------------
```

```
void Range_Config (void)
{
//          SPI_CFG(0x00,0x00);                                  // Set for sleep
            SPI_CFG(0x00,0x48);                                  // Set up for Freq Synth mode
            SPI_CFG(0x01,0xAC);                          // Set to Buffered Mode,Enable PKT handling
            SPI_CFG(0x02,0x07);                                  // Set Freq dev = 50kHz
            SPI_CFG(0x03,0x07);                                  // Data Rate = 2000bps
//          SPI_CFG(0x04,0x00);                                  // Default
            SPI_CFG(0x05,0xCF);                  // Set FIFO depth to 64 bytes,RXFIFO int=64 bytes
            SPI_CFG(0x06,0x5F);                                  // Set freq1 to 916.5 MHz
            SPI_CFG(0x07,0x50);                                  //                      "
            SPI_CFG(0x08,0x23);                                  //                      "
//          SPI_CFG(0x09,0x00);                                  // Default
//          SPI_CFG(0x0A,0x00);                                  // Default
//          SPI_CFG(0x0B,0x00);                                  // Default
            SPI_CFG(0x0C,0x00);                          // Set PA rise/fall to 3usec,low rX current
            SPI_CFG(0x0D,0x08);                                  // IRQ setups
            SPI_CFG(0x0E,0x11);                                  // TX on 1st wrt,PLL lock on pin 23
//          SPI_CFG(0x0F,0x00);                                  // Default
            SPI_CFG(0x10,0xA3);                                  // Set BWFilt
            SPI_CFG(0x11,0x38);                                  // Set PolyFilt
            SPI_CFG(0x12,0x38);                          // Set DCLK dis,Bitsych ON,Pat Recog en
            SPI_CFG(0x13,0x07);                                  // Default
//          SPI_CFG(0x14,0x00);                                  // Default
//          SPI_CFG(0x15,0x00);                                  // Default
            SPI_CFG(0x16,0xE2);                                  // Set Synch pattern3
            SPI_CFG(0x17,0xE2);                                  // Set Synch pattern2
            SPI_CFG(0x18,0xE2);                                  // Set Synch pattern1
            SPI_CFG(0x19,0xE2);                                  // Set Synch pattern0
            SPI_CFG(0x1A,0x70);                                  // Set TxInterpFilt,max TX PWR
            SPI_CFG(0x1B,0x00);                                  // ClkOut dis
            SPI_CFG(0x1C,0x7F);                                  // Set Pkt length to 64 bytes
//          SPI_CFG(0x1D,0x00);                                  // Default
            SPI_CFG(0x1E,0xE8);                          // Var lngth mode,4 byte Preamb,Whiten Off,CRC On
            SPI_CFG(0x1F,0x00);                                  // Clear FIFO if CRC fails

//********* INIT HOPPER *******
            hop_channel = 0;
            current_reg = 2;

            SPI_CFG(0x06,freq_hop[hop_channel][0]);          // load first reg1
            SPI_CFG(0x07,freq_hop[hop_channel][1]);          // load scond reg1
            SPI_CFG(0x08,freq_hop[hop_channel][2]);          // load therd reg1
            Hop();
//******************************

            if(!PLLCK)
            {
                        ComLED = 1;
                        while(PLLCK == 0)
                        {
                        }
            }
            ComLED = 1;
            mSecDly(50);
            ComLED = 0;


}

//-------------------------------------------------------------------------------------
// Range_TX: Configures the device for TX mode and sends the Range Test packet.
```

```
// After it is sent, the device will configure to RX mode to verify CRC and receive
// the acknowledge if CRC passed.
//-------------------------------------------------------------------------------------
void Range_TX (void)
{
        uchar len = 0;
        uchar *pMem;                                    // pointer to access memory

        MODE=1;                                         // TX mode configure
        len = strlen(rangeTestStr);                     // Get length of string
        pMem = rangeTestStr;                            // Assign addr of string location
        Do_TX(pMem,len);
        EA = 1;                                         // Enable interrupts

}

//-------------------------------------------------------------------------------------
// Transmit Data: This routine passes in the pointer and length of the string to send
// and configures the device for transmit mode. The data is then written to the FIFO
// via the SPI and is sent over the air.
//-------------------------------------------------------------------------------------
void Do_TX(uchar *pMem, uchar len)
{

        EA = 0;                                         // Temporarily disable interrupts
        MODE=1;                                         // TX mode configure
        TXMD=1;                                         // Turn on TX LED
        RXMD=0;                                         // Turn off RX LED

        SetforTx();                                     // Set for TX mode
        SPI_CFG(0x0D,0x08);                             // IRQ setups
        mSecDly(1);                                     // Delay for switchover time

        SSDAT=0;                                        // Select Data
        SPI0DAT = len;                                  // Write string length to SPI for var len mode
        Wait_for_SPIF();
        SSDAT = 1;                                      // Deselect Data

        while(*pMem)                                    // while not null character
        {
                SSDAT=0;                                // Select Data
                SPI0DAT=*pMem;          // Write data back-to-back to SPI to be written into FIFO for TX
                Wait_for_SPIF();
                SSDAT=1;                                // Deselect Data
                pMem++;                                 // Increment pointer to get next byte
        }

        while(!IRQ1);                                   // Wait for TX_done

//      tmp_buf = Read_Reg(0x00);                       // Read contents of register in TRC103
//      tmp_buf = ((tmp_buf & 0x1F) | 0x60);            // Set up for RX mode,keep other settings
//      SPI_CFG(0x00,tmp_buf);                          // write SPI
//      SPI_CFG(0x0D,0x08);                             // restore IRQ setups

}

//-------------------------------------------------------------------------------------
// Acknowledge Routine: Sends an Ack to the originator identifying a good received packet
//-------------------------------------------------------------------------------------
void Ack(void)
{
        uchar len = 0;
```

```
	uchar *pMem;				// pointer to access memory

	len = strlen(AckStr);			// Get length of string
	pMem = AckStr;				// Assign addr of string location
	Do_TX(pMem,len);
	SetforRx();				// Set for RX mode
	SPI_CFG(0x0D,0x08);			// restore IRQ setups
	EA = 1;					// Enable interrupts

}

//-------------------------------------------------------------------------------------
// Not Acknowledge Routine: Sends a nAck to the originator identifying a failed packet receive
//-------------------------------------------------------------------------------------
void nAck(void)
{
	uchar len = 0;
	uchar *pMem;				// pointer to access memory

	len = strlen(nAckStr);			// Get length of string
	pMem = nAckStr;				// Assign addr of string location
	Do_TX(pMem, len);
	SetforRx();				// Set for RX mode
	SPI_CFG(0x0D,0x08);			// restore IRQ setups
	EA = 1;					// Enable interrupts

}

//-------------------------------------------------------------------------------------
// Range_RX: Configure device for Receive Range Test mode.
// This routine will configure the device for Buffered RX mode.
// After receiving the CRC passed IRQ, the data will be read out
// of the FIFO to verify correct msg.
// After verification, the device will configure to TX mode and
// send an acknowledge (ACK) or not acknowledge (NACK).
//-------------------------------------------------------------------------------------
void Range_RX (void)
{

	MODE=0;					// RX mode configure
	TXMD=0;					// Turn off TX LED
	RXMD=1;					// Turn on RX LED
	SetforRx();				// Set for RX mode
	SPI_CFG(0x0D,0x08);			// IRQ setups
//	SPI_CFG(0x03,0x63);			// Data Rate = 2000bps
	mSecDly(1);				// Delay for switchover time
}

//-------------------------------------------------------------------------------------
// Config_SLP: Configure device for SLEEP mode.
// Address Reg for writing Config info:
//
//				START(0) | R/nW | A4 | A3 | A2 | A1 | A0 | STOP(0)
//
//	READ = 0x40-0x7E
//	WRITE = 0x00-0x3E
//-------------------------------------------------------------------------------------
void Config_SLP (void)
{
	uchar tmp_buf;

	MODE=2;					// Sleep mode configure
```

```
	TXMD=0;                                 // Turn off TX LED
	RXMD=0;                                 // Turn off RX LED
	tmp_buf = Read_Reg(0x00);               // Read contents of register in TRC103
	tmp_buf = ((tmp_buf & 0x1F) | 0x00);    // Set up for Sleep mode,keep other settings
	SPI_CFG(0x00,tmp_buf);                  // write SPI

}

//-------------------------------------------------------------------------------------
// SPI_CFG: Routine to write and/or read using SPI
//
//                                  START(0) | R/nW | A4 | A3 | A2 | A1 | A0 | STOP(0)
//
// The address must be shifted left one bit for proper positioning.
//
//      READ = 0x40-0x7E
//      WRITE = 0x00-0x3E
//
//-------------------------------------------------------------------------------------
void SPI_CFG (uchar addr,uchar dat)
{
	CmdLED = 1;                             // LED On
	EA = 0;                                 // disable interrupts temporarily
	SSCFG=0;                                // Select Config
	SPI0DAT = addr << 1;                    // Shift addr left one bit and send
	Wait_for_SPIF();                        // Wait for SPI to finish
	SPI0DAT = dat;                          // Write Config DATA to SPI
	Wait_for_SPIF();                        // Wait for SPI to finish
	SSCFG=1;                                // DeSelect Config
	mSecDly(5);
	EA = 1;                                 // re-enable interrupts
	CmdLED = 0;                             // LED Off

}

//-------------------------------------------------------------------------
// mSecDly: Routine to give 1msec delay (2457 loops = 1msec)
//-------------------------------------------------------------------------
// mSecDlyCnt is passed from the calling routine and is equal to the
// number of msec delay.
//-------------------------------------------------------------------------
void mSecDly (uint mSecDlyCnt)
{
	uint i;                                 // Timer Count Reg

	while(mSecDlyCnt > 0)
	{
		mSecDlyCnt--;
		for(i=2457;i>0;i--);            // Gives exactly 1msec delay (571usec/tick)
	}

}

//-------------------------------------------------------------------------
// Verify: Routine to load contents of FIFO into buffer and perform
// a compare against the range test string to verify what was RX'd.
// If verification passes then flash the Green LED.
//-------------------------------------------------------------------------
/*void Verify(void)
{
	uchar x;
	uchar *pBuf;
```

```
        pBuf = Buffer;
        SPI_CFG(0x0D,0x08);                     // Map IRQ0 to nFIFOEMPY signal

        while(IRQ0 == 1)                        // Test if FIFO empty
        {

                SSDAT = 0;
                SPI0DAT = 0x00;                 // Send dummy byte
                while(!SPIF)
                {
                                                // do nothing,Wait for SPI done
                }
                SSDAT = 1;
                SPIF = 0;                       // Reset flag
                *pBuf = SPI0DAT;                // Move FIFO byte into memory
                pBuf++;                         // increment pointer
        }

        x = strcmp(Buffer,rangeTestStr);

        if(x == 0)
        {
                GoodLED = 1;
                mSecDly(50);
                GoodLED = 0;
        }
        else
        {
                ErrLED = 1;
                mSecDly(50);
                ErrLED = 0;
        }

        SPI_CFG(0x0D,0x00);                     // re-Map IRQ0

}
*/

//-------------------------------------------------------------------------
// Read: Routine to load contents of FIFO into buffer for verification.
//-------------------------------------------------------------------------
void Read(void)
{
        uchar *pBuf;
        uchar byt_cnt;

        pBuf = Buffer;
        SPI_CFG(0x0D,0x88);                     // Map IRQ0 to nFIFOEMPY signal
        SSDAT = 0;
        SPI0DAT = 0x00;                         // Send dummy byte
        Wait_for_SPIF();                        // Wait for SPI to finish
        *pBuf = SPI0DAT;                        // Move FIFO byte into memory
        byt_cnt = *pBuf;                        // Get payload byte count
        SSDAT = 1;
        byt_cnt--;

        while(IRQ0 && byt_cnt)                  // Test if FIFO empty and count=0
        {
                SSDAT = 0;
                SPI0DAT = 0x00;                 // Send dummy byte
                Wait_for_SPIF();                // Wait for SPI to finish
```

```
                *pBuf = SPI0DAT;                // Move FIFO byte into memory
                SBUF0 = SPI0DAT;                // Send to terminal via 232
                SSDAT = 1;
                Wait_for_UART_Tx();
                byt_cnt--;
                pBuf++;
        }
                SBUF0 = 0x0D;                   // Send CR
                Wait_for_UART_Tx();
                SBUF0 = 0x0A;                   // Send LF
                Wait_for_UART_Tx();
                SPI_CFG(0x0D,0x08);             // re-Map IRQ0
}

//-----------------------------------------------------------------------
// Tmr0: Routine to initialize and start Timer0
//-----------------------------------------------------------------------
// This is used as a general purpose timer, but typically for
// a timeout interval when looking for data so that the processor
// does not get into an infinite loop looking for data.
//-----------------------------------------------------------------------
void Tmr0 (uchar rTH0,uchar rTL0)
{
        TH0 = rTH0;                     // Set High Byte
        TL0 = rTL0;                     // Set Low Byte
        TR0 = 1;                        // Enable Timer
}

//-----------------------------------------------------------------------
// External Interrupt 0 ISR (TX/RX/SLP MODE CHANGE)
//-----------------------------------------------------------------------
// (0x0003h)
// When the button (PB1) is pressed, it enters a debounce time and rechecks the
// state of the button to make sure it is still pressed and is not a glitch.
// This interrupt changes the mode of operation of the board to TX, RX, or Sleep
// when the button is pressed, as well as the LED color:
//              TX - Red
//              RX - Green
//              SLEEP - No color (all off)
//-----------------------------------------------------------------------

void MODECHG_ISR (void) interrupt 0
{

        EA=0;                                                   // Disable Interrupts temporarily
//      mSecDly(75);                                            // Debounce delay 75msec

        if(PB1==0)
        {
                if(MODE==0 && fRTESTx==0 && fRTESTRx==0)
                {
                        MODE=1;
                                                                // TX mode configure
                        TXMD=1;
                                                                // Turn on TX LED
                        RXMD=0;
                                                                // Turn off RX LED
                        Config_TX();                            // Do TX config routine
                }

                else if(MODE==1 && fRTESTx==0 && fRTESTRx==0)
                {
```

```
                        MODE=2;
        // Sleep mode configure
                        TXMD=0;
        // Turn off TX LED
                        RXMD=0;
        // Turn off RX LED
                        Config_SLP();                   // Do Sleep config routine
                }

                else if(MODE==2 && fRTESTx==0 && fRTESTRx==0)
                {
                        MODE=0;
        // RX mode configure
                        TXMD=0;
        // Turn off TX LED
                        RXMD=1;
        // Turn on RX LED
                        Config_RX();                    // Do RX config routine
                }

//      mSecDly(75);                                    // Debounce delay 75msec
        EA=1;                                           // Enable Interrupts

        }

}

//-----------------------------------------------------------------------
// External Interrupt 1 ISR (RANGE TEST ON/OFF)
//-----------------------------------------------------------------------
// (0x0013h)
// When the button (PB2) is pressed, it enters a debounce time and rechecks the state
// of the button to make sure it is still pressed and is not a glitch.
// This interrupt enables or disables the Range Test function, as well as the
// LED:
//              Range Test ON - Red
//              Range Test OFF - Green
//-----------------------------------------------------------------------

void RANGETST_ISR (void) interrupt 2
{
        uint    j = 0;                                  // msec counter

        EA=0;                                           // Disable Interrupts temporarily
        mSecDly(50);                                    // Debounce delay

        while(PB2 == 0 && j < 505)
        {
                j++;                                    // Increment Button-hold-down counter
                mSecDly(1);                             // Do 50 msec delay and then retest if button
                                                        // still pressed.
        }

        if(j > 500)
        {
                if(fRTESTx == Stop)
                {
                        fRTESTx = Start;                // Range Test ON
                        RTMD = 1;                       // LED on
                }

                else
```

```
                {
                        fRTESTx = Stop;                         // Range Test OFF
                        RTMD = 0;                               // LED Off
                }

        }
        else if(j < 500 && RTMD == 0)
        {
                        RTMD=1;                                 // LED on
                        fRTESTRx = Start;                       // Set for Range Msg receive
                        TXMD=0;                                 // Turn off TX LED
                        RXMD=1;                                 // Turn on RX LED
                        fTermAct = 1;                           // Next select for terminal

        }
        else if(RTMD == 1 && fTermAct == 1)
        {
                        fRTESTRx = Stop;                        // Set for Range Msg receive
                        MODE=0;                                 // RX mode configure
                        RTMD=0;                                 // LED off
                        mSecDly(75);
                        RTMD=1;                                 // LED on
                        mSecDly(75);
                        RTMD=0;                                 // LED off
                        mSecDly(75);
                        RTMD=1;                                 // LED on
                        fTermAct = 0;                           // Next select for terminal (off)
                        fTermDat = 1;                           // Terminal Active

        }
        else if(RTMD == 1 && fTermAct == 0)
        {
                        fRTESTx = Stop;                         // Range Test OFF
                        MODE=0;                                 // RX mode configure
                        TXMD=0;                                 // Turn off TX LED
                        RXMD=1;                                 // Turn on RX LED
                        Config_RX();                            // Do RX config routine
                        RTMD=0;                                 // LED off
                        fTermDat = 0;                           // Terminal Inactive

        }

        mSecDly(25);                                            // delay
        EA = 1;                                                 // Enable Interrupts

/*
//****************************************
// Used to put a 1kHz signal on DAT pin
//****************************************
        P2MDOUT   = 0x8B;                                       // **make 2.3 output
        if(MODE == 1)
        {
        while(PB2==0)
        {
                DAT = 1;
                for(j=1230;j>0;j--);                    // Gives exactly 1msec delay (571usec/tick)
                DAT=0;
                for(j=870;j>0;j--);                     // Gives exactly 1msec delay (571usec/tick)
        }
           P2MDOUT   = 0x83;                                    // **make 2.3 input
        }
*/
```

```
} // End RANGETST_ISR

//-------------------------------------------------------------------------
//              UART ISR (0x0023h)
//-------------------------------------------------------------------------

void UART_ISR (void) interrupt 4
{
        uchar cntr = 0x00;
        uchar datCntr = 0x00;
        uchar byte1,byte2;
        uchar tmp_buf;
        uchar cfgAddrH;                         // Address Reg
        uchar cfgAddrL;                         // Address Reg
        uchar cfgDatH;                          // DATA Reg
        uchar cfgDatL;                          // DATA Reg
        uchar len;                              // length of string
        uchar len2;                             // length of 2nd string if > 64 bytes

        EA = 0;                                 // Disable Interrupts temporarily

        RI0 = 0;                                // Clear RX interrupt
//      TI0 = 0;                                // Clear TX interrupt

        if (SBUF0 == 'R' || SBUF0 == 'r')
        {
                Wait_for_UART_Rx();             // Wait for UART to receive nxt byte

                if (SBUF0 == 0x0D)
                {

                        SSCFG = 0;
                        while(cntr != 0x20)
                        {
                                ComLED = 1;
                                SPI0DAT = ((0x20 | cntr) << 1);
                                Wait_for_SPIF();                        // Wait for SPI to finish
                                SPI0DAT = ((0x20 | cntr) << 1);
                                Wait_for_SPIF();                        // Wait for SPI to finish
                                tmp_buf = SPI0DAT;

                                HEX2ASC (cntr, &byte1, &byte2);
                                SBUF0 = byte2;
                                Wait_for_UART_Tx();             // Wait for UART Tx to finish
                                SBUF0 = byte1;
                                HEX2ASC (tmp_buf, &byte1, &byte2);
                                Wait_for_UART_Tx();             // Wait for UART Tx to finish
                                SBUF0 = byte2;
                                Wait_for_UART_Tx();             // Wait for UART Tx to finish
                                SBUF0 = byte1;
                                Wait_for_UART_Tx();             // Wait for UART Tx to finish
                                SBUF0 = 0x0D;
                                Wait_for_UART_Tx();             // Wait for UART Tx to finish
                                SBUF0 = 0x0A;
                                Wait_for_UART_Tx();             // Wait for UART Tx to finish
                                cntr++;
                        }

                        SSCFG = 1;
                        ComLED = 0;
                }
```

```
}

if (SBUF0 == 'S' || SBUF0 == 's')
{
        Wait_for_UART_Rx();                     // Wait for UART to receive nxt byte

        if (SBUF0 == 'W' || SBUF0 == 'w')
        {

                ComLED = 1;
                Wait_for_UART_Rx();             // Wait for UART to receive nxt byte

                cfgAddrH = ASC2HEX (SBUF0);     // Temp Store address

                Wait_for_UART_Rx();             // Wait for UART to receive nxt byte

                cfgAddrL = ASC2HEX (SBUF0);     // Temp Store address
                *pBuf = (cfgAddrH << 4) | cfgAddrL; // 'Or' values to make hex value and store
                pBuf++;
                                                // Increment pointer
                cfgcnt++;                       // ement pointer

                Wait_for_UART_Rx();             // Wait for UART to receive nxt byte

                cfgDatH = ASC2HEX (SBUF0);      // Temp Store address

                Wait_for_UART_Rx();             // Wait for UART to receive nxt byte

                cfgDatL = ASC2HEX (SBUF0);      // Temp Store address
                tmp_buf = (cfgDatH << 4) | cfgDatL;
                *pBuf = (cfgDatH << 4) | cfgDatL;   // 'Or' values to make hex value
                pBuf++;
                                                // Increment pointer
                cfgcnt++;                       // Increment pointer

                if(cfgcnt > 0x3D)
                {
                        fUpdate = Yes;          // set flag to write data to SPI device
                        cfgcnt = 0;             // re-initialize count
                }
                ComLED = 0;

        }
}

if (SBUF0 == 'W' || SBUF0 == 'w')
{

        ComLED = 1;
        Wait_for_UART_Rx();             // Wait for UART to receive nxt byte

        cfgAddrH = ASC2HEX (SBUF0);     // Temp Store address

        Wait_for_UART_Rx();             // Wait for UART to receive nxt byte

        cfgAddrL = ASC2HEX (SBUF0);     // Temp Store address
        byte1 = (cfgAddrH << 4) | cfgAddrL; // 'Or' values to make hex value and store

        Wait_for_UART_Rx();             // Wait for UART to receive nxt byte

        cfgDatH = ASC2HEX (SBUF0);      // Temp Store address
```

```
                Wait_for_UART_Rx();             // Wait for UART to receive nxt byte

                cfgDatL = ASC2HEX (SBUF0);      // Temp Store address
                byte2 = (cfgDatH << 4) | cfgDatL;   // 'Or' values to make hex value
                SPI_CFG(byte1,byte2);
                ComLED = 0;


        }

        if(SBUF0 == 0x02)                   // Look for start of text
        {
                pBuf = Buffer;                                  // get address of buffer

                while(SBUF0 != 0x03 && datCntr <= 128)      // Look for end of text(0x03) or full buffer
                {
                        Wait_for_UART_Rx();                     // Wait for UART to receive nxt byte

                        *pBuf = SBUF0;
                        pBuf++;
                        datCntr++;
                }
                datCntr--;                                      // back up one byte to remove
0x03EOT marker
                TXMD=1;                                         // Turn on TX LED
                RXMD=0;                                         // Turn off RX LED
                Range_Config();
                SetforTx();                                     // Set for TX mode
                SPI_CFG(0x0D,0x08);                             // IRQ setups(RX0-pyldrdy,RX1-
CRC,TX1-TXstop)
                pBuf = Buffer;
                len = datCntr;                          //***strlen(Buffer);// Get length of string
                mSecDly(3);                                     // delay for PLL/synth locck
                if(len > 64)
                {
                        len2 = datCntr - 64;
                        len = 64;
                        SSDAT = 0;                                      // Select Data
                        SPI0DAT = len;    // Write data back-to-back to SPI to be written into FIFO for TX
                        Wait_for_SPIF();                                // Wait for SPI to finish
                        SSDAT = 1;                                      // Deselect Data
                        len--;

                        while(len)                                      // If data cntr != 0
                        {
                                SSDAT = 0;                              // Select Data
                                SPI0DAT = *pBuf;            // Write data back-to-back to SPI to be written
                                Wait_for_SPIF();                // Wait for SPI to finish
                                SSDAT = 1;                      // Deselect Data
                                len--;                          // Decrement data counter
                                pBuf++;                         // Increment buffer cntr
                        }

                        while(IRQ0);                            // loop until FIFO empty //
                        while(!IRQ1);                           // Wait for TX stopped (active high)

                        SSDAT = 0;                              // Select Data
                        SPI0DAT = len2;   // Write data back-to-back to SPI to be written into FIFO for TX
                        Wait_for_SPIF();                        // Wait for SPI to finish
                        SSDAT = 1;                              // Deselect Data
                        len2--;
```

```
                    while(len2)                                     // If data cntr != 0
                    {
                        SSDAT = 0;                          // Select Data
                        SPI0DAT = *pBuf;                    // Write data back-to-back to SPI to be
                        Wait_for_SPIF();                    // Wait for SPI to finish
                        SSDAT = 1;                          // Deselect Data
                        len2--;                             // Decrement data counter
                        pBuf++;                             // Increment buffer cntr
                    }

                    while(IRQ0);                            // wait for FIFO empty //
                    while(!IRQ1);                           // Wait for TX stopped (active high)

                    TXMD=0;                                 // Turn off TX LED
                    RXMD=1;                                 // Turn on RX LED
                    SetforRx();                             // Set for RX mode
                    SPI_CFG(0x0D,0x08);                     // IRQ setups(RX0-pyldrdy,RX1-
                    mSecDly(1);                             // delay for PLL/synth lock
                    fWaitforAck = 1;            // set flag so that you dont ack an ack and ping pong

                }

                else

                    SPI_CFG(0x0D,0x00);                             // IRQ1-FIFO Full (RX0-)
                    SSDAT = 0;                                      // Select Data
                    SPI0DAT = len;                                  // Write data back-to-back to
                    Wait_for_SPIF();                                // Wait for SPI to finish
                    SSDAT = 1;                                      // Deselect Data
                    len--;                                          // Decrement data counter

                    while(len)                                      // If data cntr != 0
                    {
                        while(!IRQ1 && len)                 // If FIFO not Full and cntr not 0
                        {
                            SSDAT=0;                                // Select Data
                            SPI0DAT=*pBuf;                          // Write data back-to-back to
                            Wait_for_SPIF();                        // Wait for SPI to finish
                            SSDAT=1;                                // Deselect Data
                            len--;                                  // Decrement data counter
                            pBuf++;                                 // Increment buffer cntr
                        }
                    }

                    SPI_CFG(0x0D,0x08);         // IRQ1-FIFO Full (RX0-pyldrdy,RX1-CRC,TX1-txstop)
                    while(IRQ0);                                    // wait for FIFO empty)
                    while(!IRQ1);                                   // Wait for TX stopped

                    TXMD=0;                                         // Turn off TX LED
                    RXMD=1;                                         // Turn on RX LED
                    //SPI_CFG(0x00,0x68);                           // Set for RX
                    SetforRx();                                     // Set for RX mode
                    SPI_CFG(0x0D,0x08);                             // IRQ setups
                    mSecDly(1);                                     // delay for PLL/synth lock
                    fWaitforAck = 1;            // set flag so that you dont ack an ack and ping pong

                }
            }

            EA=1;                                                   // Enable Interrupts
}
```

```
//-------------------------------------------------------------------------
//              Timer 3 Interrupt ISR (Master timer)
//-------------------------------------------------------------------------
// (0x0073h)
// This is used as a master timer for periodic funtions like:
//                      -Initiating ADC conversion for potentiometer
//                      -Initiating ADC conversion for (internal) Temp Sensor
//-------------------------------------------------------------------------

void Master_Time (void) interrupt 14
{
   TMR3CN &= 0x3F;                                                  // clear interrupt
              t3cnt++;                                              // increment counter
              hhtime++;
              mhtime++;

}

//-------------------------------------------------------------------------
//              ASCII2HEX conversion
//-------------------------------------------------------------------------

uchar ASC2HEX (uchar byte)
{
              uchar hex;

              if(!(byte & 0xC0))
              {
                            hex = byte & 0x0F;
              }
              else if(!(byte & 0xB0) || !(byte & 0x90))
              {
                            hex = 9+(byte & 0x0F);
              }

              return hex;
}

//-------------------------------------------------------------------------
//              HEX2ASCII conversion
//-------------------------------------------------------------------------

void HEX2ASC (uchar byte,uchar *pByte1,uchar *pByte2)
{
              uchar tmp1,tmp2;

              tmp1 = (byte & 0x0F);
              tmp2 = (byte & 0xF0) >> 4;

              if(tmp1 <= 0x09)
              {
                            *pByte1 = 0x30 | tmp1;
              }
              else if((tmp1 > 0x09) || !(tmp1 & 0x90))
              {
                            *pByte1 = (0x40 | tmp1) - 9;
              }

              if(tmp2 <= 0x09)
              {
                            *pByte2 = 0x30 | tmp2;
              }
```

```
        else if((tmp2 > 0x09) || !(tmp2 & 0x90))
        {
                *pByte2 = (0x40 | tmp2) - 9;
        }

}

//---------------------------------------------------------------------------
// Wait_for_SPIF: Wait for the SPI to finish sending byte
//---------------------------------------------------------------------------
void Wait_for_SPIF(void)
{
        while(!SPIF);
        SPIF = 0;
}

//---------------------------------------------------------------------------
// Wait_for_UART_Tx: Wait for the UART to finish sending byte
//---------------------------------------------------------------------------
void Wait_for_UART_Tx(void)
{
        while(!TI0);
        TI0 = 0;
}

//---------------------------------------------------------------------------
// Wait_for_UART_Rx: Wait for the UART to receive next byte
//---------------------------------------------------------------------------
void Wait_for_UART_Rx(void)
{
        while(!RI0);
        RI0=0;                                          // Clear interrupt
}

//---------------------------------------------------------------------------
// Read_Reg: Read the current value of a register.  Pass in addr, pass out val
//---------------------------------------------------------------------------
uchar Read_Reg(uchar addr)
{
        uchar tmp_buf2;

        SSCFG = 0;
        SPI0DAT = ((0x20 | addr) << 1);
        Wait_for_SPIF();                                // Wait for SPI to finish
        SPI0DAT = ((0x20 | addr) << 1);
        Wait_for_SPIF();                                // Wait for SPI to finish
        tmp_buf2 = SPI0DAT;
        SSCFG = 1;

        return tmp_buf2;

}

//---------------------------------------------------------------------------
// Set for RX: sets the TRC103 into RX mode and retains the other settings
//---------------------------------------------------------------------------
void SetforRx(void)
{
        uchar tmp_buf;

        tmp_buf = Read_Reg(0x00);                       // Read contents of register in TRC103
        tmp_buf = ((tmp_buf & 0x1F) | 0x60);            // Set up for RX mode,keep other settings
```

```
	SPI_CFG(0x00,tmp_buf);                                  // write SPI
}

//-------------------------------------------------------------------------
// Set for TX: sets the TRC103 into TX mode and retains the other settings
//-------------------------------------------------------------------------
void SetforTx(void)
{
	uchar tmp_buf;

	tmp_buf = Read_Reg(0x00);                               // Read contents of register in TRC103
	tmp_buf = ((tmp_buf & 0x1F) | 0x80);                    // Set up for TX mode,keep other settings
	SPI_CFG(0x00,tmp_buf);                                  // write SPI
}



void Rst_CRC(void)
{
	uchar tmp_buf;

	tmp_buf = Read_Reg(0x1E);                               // Read contents of register in TRC103
	tmp_buf = (tmp_buf | 0x01);                             // Reset CRC
	SPI_CFG(0x1E,tmp_buf);                                  // write SPI
}



void Clear_FIFO(void)
{
	uchar tmp_buf;

	tmp_buf = Read_Reg(0x0D);                               // Read contents of register in TRC103
	tmp_buf = (tmp_buf | 0x01);                             // Clear FIFO
	SPI_CFG(0x0D,tmp_buf);                                  // write SPI
}




void Hop(void)
	// Change channels and reg's
{
		// Toggle between reg1 and reg2


	if (current_reg == 1)

		{
			SPI_CFG(0x00,0x69);
	// Reg 2 active
			SPI_CFG(0x06,freq_hop[hop_channel][0]);     // load first reg1
			SPI_CFG(0x07,freq_hop[hop_channel][1]);     // load scond reg1
			SPI_CFG(0x08,freq_hop[hop_channel][2]);     // load therd reg1
			current_reg = 2;                            // do second freq reg next time
		}
	else if(current_reg == 2)
		{
			SPI_CFG(0x00,0x68);
	// reg 1 active
```

```
                    SPI_CFG(0x09,freq_hop[hop_channel][0]);
                    SPI_CFG(0x0A,freq_hop[hop_channel][1]);
                    SPI_CFG(0x0B,freq_hop[hop_channel][2]);
                    current_reg = 1;
                }
        if (hop_channel++ >= 24) hop_channel = 0;       // max 25 channels
}
```