```
/*  Code for Production 3_2_15
 <Patriot_507_Alpha_Rev01, Basic Software to operate a 2 band SSB/CW QRP Transceiver.
 See PROJECT PATRIOT SSB/CW QRP below>
 This library is free software; you can redistribute it and/or
 modify it under the terms of the GNU Lesser General Public
 License as published by the Free Software Foundation; either
 version 2.1 of the License, or (at your option) any later version.

 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 Lesser General Public License for more details.

 You should have received a copy of the GNU Lesser General Public
 License along with this library; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
 */
//  https://groups.yahoo.com/neo/groups/TenTec507Patriot/info/
// !! Disclaimer !!  !! Disclaimer !!  !! Disclaimer !!  !! Disclaimer !!  !! Disclaimer !!
//  Attention ****  Ten-Tec Inc. is not responsile for any modification of Code
//  below. If code modification is made, make a backup of the original code.
//  If your new code does not work properly reload the factory code to start over again.
//  You are responsible for the code modifications you make yourself. And Ten-Tec Inc.
//  Assumes NO libility for code modification. Ten-Tec Inc. also cannot help you with any
//  of your new code. There are several forums online to help with coding for the ChipKit UNO32.
//  If you have unexpected results after writing and programming of your modified code.
//  Reload the factory code to see if the issues are still present. Before contacting Ten_Tec Inc.
//  Again TenTec Inc. NOT RESPONSIBLE for modified code and cannot help with the rewriting of the
//  factory code!
/*
```

```
/*********  PROJECT PATRIOT SSB/CW QRP  ****************************
 * Program for the ChipKit Uno32
 * This is a simple program to demonstrate a 2 band QRP Amateur Radio Transceiver
 * Amateur Programmer Bill Curb (WA4CDM).
 * This program will need to be cleaned up a bit and no doubt be made more efficient!
 * Compiled using the MPIDE for the ChipKit Uno32.
 *
 * Prog for ad9834
 * Serial timming setup for AD9834 DDS
 * start > Fsync is high (1), Sclk taken high (1), Data is stable (0, or 1),
 * Fsync is taken low (0), Sclk is taken low (0), then high (1), data changes
 * Sclk starts again.
 * Control Register D15, D14 = 00, D13(B28) = 1, D12(HLB) = X,
 * Reset goes high to set the internal reg to 0 and sets the output to midscale.
 * Reset is then taken low to enable output.
 ************************************************
 * This is real basic code to get things working.
 ***************************************************************
 * The pinout for the LCD is as follows: Also the LCD is setup up for 4 lines 20 charactors.
 * LCD RS pin to digital pin 26
 * LCD Enable pin to digital pin 27
 * LCD D4 pin to digital pin 28
 * LCD D5 pin to digital pin 29
 * LCD D6 pin to digital pin 30
 * LCD D7 pin to digital pin 31
 * LCD R/W pin to ground
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)    analogWrite(Side_Tone, 127);
 ***************************************************************
 * SELECT button steps from in
 * BW ( <Wide, green>, <Medium, yellow>, <Narrow, red> ).
 * STEP ( <100 hz, green, <1Khz, yellow>, 10Khz, red> ).
 * BND ( < 40M >, < 20M >, < , > ) OTHER has yet to be defined
 *
 * Default Band_width will be wide ( Green led lite ).
 * When pressing the function button one of three leds will lite.
 * as explained above the select button will choose which setting will be used.
 * The Orange led in the Ten-Tec logo will flash to each step the STEP is set
 * too when tuning.  As it will also turn on when at the BAND edges.
 * The TT logo led will also flash to indicate ALC. Input levels should be kept low enough
 * to only flash this led on Peaks.
 * Default frequency on power up will be the calling frequency of the
 * 40 meter band.
 * I.F. Frequency used is 9.0 mhz.
 * DDS Range is:
 * 40 meters will use HI side injection.
 * 9(I.F.) + 7(40m) = 16mhz.  9(I.F.) + 7.30 = 16.3 mhz.
 * 20 meters will use LO side injection.
 * 14(20m) - 9(I.F.) = 5mhz.  14.350(20m) - 9(I.F.) = 5.35 mhz.
 *
```

```
* The Headphone jack can supply a headphone or speaker. The header pins(2)
 * if shorted will drive a speaker.
 * Unshorted inserts 100 ohm resistors in series with the headphone to limit
 * the level to the headphones.
 *
 * The RIT knob will be at 0 offset in the Top Dead Center position. And will
 * go about -500 hz to +500 hz when turned to either extreme. Total range
 * about +/- 500 hz. This may change!
 *
 ********************************************************************************
 *
 * Added an MCP23017 16-bit I/O Expander with Serial Interface to free up
 * some I/O pins on the ChipKit Uno32 board.
 * The parts of the 507 being controlled by this ic will be the Multi-purpose
 * leds, the Select leds and the Wide/medium/Narrow control.
 * 5/1/2014 added a couple of routines to keep the filter wide on TX of SSB or CW
 * Port B can be used by the user for external control.
 *
 * GPAO (21) Select Green led
 * GPA1 (22) Select Yellow led
 * GPA2 (23) Select Red led
 * GPA3 (24) MP_A Green led
 * GPA4 (25) MP_B Yellow led
 * GPA5 (26) MP_C Red led
 * GPA6 (27) Medium A8 BW_control
 * GPA7 (28) Narrow A9 BW_control
 *
 * A mask function will be used to combine the various bits together.
 */
```

```
// various defines
#define SDATA_BIT                          11          //
#define SCLK_BIT                           12          //
#define FSYNC_BIT                          13          //
#define RESET_BIT                          10          //
#define FREQ_REGISTER_BIT                  9           //
#define PHASE_REGISTER_BIT                 8
#define AD9834_FREQ0_REGISTER_SELECT_BIT   0x4000      //
#define AD9834_FREQ1_REGISTER_SELECT_BIT   0x8000      //
#define FREQ0_INIT_VALUE                   0x00000000  // 0x01320000

#define led                                13
#define MUTE                               4
#define MIC_LINE_MUTE                      34

#define Side_Tone                          3           //

#define PTT_SSB                            22          // ptt input pulled high
#define SSB_CW                             42          // control line for /SSB_CW switches
                                                       // output, high for cw , low for ssb
#define TX_Dah                             33          //
#define TX_Dit                             32          //
#define TX_OUT                             38          //
#define Band_End_Flash_led                 24          // also this led will flash every
                                                       // 100/1khz/10khz is tuned
#define Band_Select                        41          // output for band select
#define Multi_Function_Button              5           //
#define Flash                              Band_End_Flash_led

#define Select_Button                      2           //

#define Wide_BW                            0           //
#define Medium_BW                          1           //
#define Narrow_BW                          2           //


#define Step_100_Hz                        0
#define Step_1000_hz                       1
#define Step_10000_hz                      2

#define  Other_1_user                      0           // 40 meters
#define  Other_2_user                      1           // 20 meters
#define  Other_3_user                      2           // anything you desire!
```

```
const int RitReadPin        = A0;  // pin that the sensor is attached to used for a rit routine
later.
int RitReadValue            = 0;
int RitFreqOffset           = 0;
int old_RitFreqOffset       = 0;

const int SmeterReadPin     = A1;  // To give a realitive signal strength based on AGC voltage.
int SmeterReadValue         = 0;

const int BatteryReadPin    = A2;  // Reads 1/5 th or 0.20 of supply voltage.
int BatteryReadValue        = 0;

const int PowerOutReadPin   = A3;  // Reads RF out voltage at Antenna.
int PowerOutReadValue       = 0;

const int CodeReadPin       = A6;  // Can be used to decode CW.
int CodeReadValue           = 0;

const int CWSpeedReadPin    = A7;  // To adjust CW speed for user written keyer.
int CWSpeedReadValue        = 0;
```

```
#include "Wire.h"
#include <LiquidCrystal.h>    //  LCD Stuff

LiquidCrystal lcd(26, 27, 28, 29, 30, 31);      //  LCD Stuff

const char txt52[5]         = " ";
const char txt57[6]         = "FREQ:" ;
const char txt60[6]         = "STEP:";
const char txt62[3]         = "RX";
const char txt63[3]         = "TX";
const char txt64[4]         = "RIT";
const char txt65[5]         = "Band";
const char txt66[4]         = "20M";
const char txt67[4]         = "40M";
const char txt69[4]         = "   ";
const char txt70[5]         = "    ";
const char txt71[6]         = "     ";
const char txt72[10]        = "        ";
const char txt85[2]         = "W";
const char txt86[2]         = "M";
const char txt87[2]         = "N";
const char txt90[5]         = "STEP";
const char txt110[4]        = "BAT";
const char txt120[3]        = "BW";
const char txt130[5]        = "MODE";
const char txt132[3]        = "CW";
const char txt135[4]        = "SSB";
const char txt140[5]        = "WIDE";
const char txt150[7]        = "MEDIUM";
const char txt160[7]        = "NARROW";
const char txt170[7]        = "      ";

String stringFREQ;
String stringREF;
String string_Frequency_Step;
String stringRIT;
String stringVolts;
String stringBW;
```

```
int TX_key;
int PTT_SSB_Key;
int old_PTT_SSB_Key;

int band_sel;                          // select band 40 or 20 meter
int band_set;
int bsm;

int Step_Select_Button          = 0;
int Step_Select_Button1         = 0;
int Step_Multi_Function_Button  = 0;
int Step_Multi_Function_Button1 = 0;

int Selected_BW                 = 0;    // current Band width
                                        // 0= wide, 1 = medium, 2= narrow
int Selected_Step               = 0;    // Current Step
int Selected_Other              = 0;    // To be used for anything

int old_bsm                     = 0;    //  this helps 5/13/14

int old_BatteryReadValue        = 0;

byte s = 0x00;                    // s = select
byte m = 0x00;                    // m = multi
byte b = 0x00;                    // b = bandwidth
byte t = 0x00;                    // s + m ored
byte old_b = 0x00;                // for the TX routine
```

```
//-----------------------------------------------------
// Encoder Stuff
const int encoder0PinA          = 6; // reversed for 507
const int encoder0PinB          = 7; // reversed for 507

int val;
int encoder0Pos                 = 0;
int encoder0PinALast            = LOW;
int n                           = LOW;

//-----------------------------------------------------
const long meter_40             = 16.285e6;      // IF + Band frequency, default for 40
                                                 // HI side injection 40 meter
                                                 // range 16 > 16.3 mhz
const long meter_20             = 5.285e6;       // Band frequency - IF, LOW  default for 20
                                                 // side injection 20 meter
                                                 // range 5 > 5.35 mhz
const long Reference            = 50.0e6;        // for ad9834 this may be
                                                 // tweaked in software to
                                                 // fine tune the Radio

long frequency_TX;
long TX_frequency;
long RIT_frequency;
long RX_frequency;
long save_rec_frequency;
long Frequency_Step;
long old_Frequency_Step;
long frequency                  = 0;
long frequency_old              = 0;
long frequency_old_TX           = 0;
long frequency_tune             = 0;
long old_frequency_tune         = 0;
long frequency_default          = 0;
long fcalc;
long IF                         = 9.00e6;         //  I.F. Frequency
long TX_Frequency               = 0;

//-----------------------------------------------------
```

```
// Debug Stuff

unsigned long   loopCount       = 0;
unsigned long   lastLoopCount   = 0;
unsigned long   loopsPerSecond  = 0;
unsigned int    printCount      = 0;

unsigned long   loopStartTime   = 0;
unsigned long   loopElapsedTime = 0;
float           loopSpeed       = 0;

unsigned long LastFreqWriteTime = 0;

void    serialDump();

//-------------------------------------------------------
```

```
void Default_frequency();
void AD9834_init();
void AD9834_reset();
void program_freq0(long freq);
void program_freq1(long freq1);
void UpdateFreq(long freq);

void RX_Rit();

//   void Frequency_up();
//   void Frequency_down();


void TX_routine();
void RX_routine();
void Encoder();
void AD9834_reset_low();
void AD9834_reset_high();

void Change_Band();
void Step_Flash();
void RIT_Read();

void Multi_Function();          //
void Step_Selection();          //
void Selection();               //
void Step_Multi_Function();     //

//-----------------------------------------------------

void clock_data_to_ad9834(unsigned int data_word);

//-----------------------------------------------------
```

```
void setup()
{
  // these pins are for the AD9834 control
  pinMode (SCLK_BIT,                OUTPUT);    // clock
  pinMode (FSYNC_BIT,               OUTPUT);    // fsync
  pinMode (SDATA_BIT,               OUTPUT);    // data
  pinMode (RESET_BIT,               OUTPUT);    // reset
  pinMode (FREQ_REGISTER_BIT,       OUTPUT);    // freq register select

  //---------------  Encoder ----------------------------
  pinMode (encoder0PinA,            INPUT);     //
  pinMode (encoder0PinB,            INPUT);     //

  //-----------------------------------------------------
  pinMode (TX_Dit,                  INPUT);     // Dit Key line
  pinMode (TX_Dah,                  INPUT);     // Dah Key line
  pinMode (TX_OUT,                  OUTPUT);    // control line for TX stuff
  pinMode (Band_End_Flash_led,      OUTPUT);    // line that flashes an led
  pinMode (PTT_SSB,                 INPUT);     // mic key has pull-up
  pinMode (SSB_CW,                  OUTPUT);    // control line for ssb cw switches

  pinMode (Multi_Function_Button,   INPUT);     // Choose from Band width, Step size, Other

  pinMode (Select_Button,           INPUT);     //  Selection from the above

  pinMode (Side_Tone,               OUTPUT);    // sidetone enable

  pinMode (Band_Select,             OUTPUT);

  pinMode (MUTE,                    OUTPUT);

  pinMode (MIC_LINE_MUTE,           OUTPUT);    // low on receive

  digitalWrite (Band_End_Flash_led, LOW);       //  not in 81324

  digitalWrite (MUTE,               LOW);


  BatteryReadValue = analogRead(BatteryReadPin);

  Default_Settings();
```

```
// I2C stuff
  Wire.begin();                             // wake up I2C bus
  Wire.beginTransmission(0x20);
  Wire.send(0x00);                          // IODIRA register
  Wire.send(0x00);                          // set all of port A to outputs
  Wire.endTransmission();
  Wire.beginTransmission(0x20);
  Wire.send(0x01);                          // IODIRB register
  Wire.send(0x00);                          // set all of port B to outputs
  Wire.endTransmission();

  //-------------------------------------------------------
  // DDS
  AD9834_init();
  AD9834_reset();                           // low to high
  //-------------------------------------------------------

  digitalWrite(TX_OUT,              LOW);   // turn off TX
  digitalWrite(SSB_CW,              LOW);   // keeps tx in ssb mode until high

  //-------------------------------------------------------
  Frequency_Step = 100;    //  Can change this whatever step size one wants
  Selected_Step = Step_100_Hz;
  DDS_Setup();
  encoder0PinALast = digitalRead(encoder0PinA);
  //attachInterrupt(encoder0PinA, Encoder, CHANGE);
  //attachInterrupt(encoder0PinB, Encoder, CHANGE);
  attachCoreTimerService(TimerOverFlow);    // See function at the bottom of the file.

  Serial.begin(115200);
  Serial.println("Patriot Ready:");

  lcd.begin(20, 4);                         // 20 chars 4 lines
                                            // or change to suit ones
                                            // lcd display

  Display_Setup();

}   //    end of setup

//=====================================================================
```

```
//====================================================================
void Display_Setup()
{
  // RX
  lcd.setCursor(0, 0);
  lcd.print(txt62);       // RX

  // RIT
  lcd.setCursor(12, 0);
  lcd.print(txt64);       // RIT

  // TX
  lcd.setCursor(0, 1);
  lcd.print(txt63);       // TX

  // BAND
  lcd.setCursor(12, 1);
  lcd.print(txt65);       // BAND
  // default band
  lcd.setCursor(17, 1);
  lcd.print(txt67);       // 40M   change this to txt66 for display of 20M

  // STEP
  lcd.setCursor(0, 2);
  lcd.print(txt90);       // STEP

  // BAT
  lcd.setCursor(12, 2);
  lcd.print(txt110);      // BAT

  // BW
  lcd.setCursor(0, 3);
  lcd.print(txt120);      // BW
  // default BW
  lcd.setCursor(3, 3);
  lcd.print(txt140);      // Wide

  // MODE
  lcd.setCursor(12, 3);
  lcd.print(txt130);      // MODE
  // default MODE
  lcd.setCursor(17, 3);
  lcd.print(txt69);
  lcd.setCursor(17, 3);
  lcd.print(txt135);      // SSB
}                         // end of Display_Setup
```

```
//=====================================================================

void Default_Settings()
{
  m = 0x08;                     //

  s = 0x01;                     //

  bsm = 0;                      //  bsm = 0 is 40 meters bsm = 1 is 20 meters

  frequency_default = meter_40; // change this to meter_20 for 20 meter default
  Default_frequency();
  b = 0x00;                     // Hardware control of I.F. filter shape wide setting
  Selected_BW = Wide_BW;

  digitalWrite (TX_OUT,               LOW);
  digitalWrite (Band_End_Flash_led,   LOW);
  digitalWrite (Side_Tone,            LOW);
  digitalWrite ( FREQ_REGISTER_BIT,   LOW);
  digitalWrite ( SSB_CW,              LOW);
  digitalWrite ( Band_Select,         LOW);
  digitalWrite ( MUTE,                HIGH);
  digitalWrite ( MIC_LINE_MUTE,       LOW);    //  receive mode
}

//---------------------------------------------------------------
void DDS_Setup()
{
  digitalWrite(FSYNC_BIT,             HIGH);   //
  digitalWrite(SCLK_BIT,              HIGH);   //
}
```

```
//======================= Main Part ================================
void loop()
{
  //  TX_routine();
  Encoder();
  TX_routine();
  RX_Rit();
  Multi_Function();

  loopCount++;
  loopElapsedTime    = millis() - loopStartTime;

  if( 1000 <= loopElapsedTime )
  {
    serialDump();    // comment this out to remove the one second tick
  }
}    //  END LOOP
```

```
//-----------------------------------------------------
void  RX_Rit()
{
  RIT_Read();
  frequency_tune  = frequency + RitFreqOffset;   // RitFreqOffset is from Rit_Read();
  UpdateFreq(frequency_tune);
  splash_RX_freq();            // this only needs to be updated when encoder changed.
}

//-----------------------------------------------------
void RIT_Read()
{
  int RitReadValueNew =0 ;
  RitReadValueNew = analogRead(RitReadPin);

  // Lowpass filter possible display role if changed
  RitReadValue = (RitReadValueNew + (12 * RitReadValue)) / 13;

  if(RitReadValue < 500)
    RitFreqOffset = RitReadValue-500;
  else if(RitReadValue < 523)
    RitFreqOffset = 0;                  // Deadband in middle of pot
  else
    RitFreqOffset = RitReadValue - 523;

  splash_RIT();                         //comment out if display is not needed
}

//-----------------------------------------------------
void UpdateFreq(long freq)
{
  if (LastFreqWriteTime != 0)
  {
    if ((millis() - LastFreqWriteTime) < 100) return;
  }
  LastFreqWriteTime = millis();
  if(freq == frequency_old) return;
  program_freq0( freq );
  frequency_old = freq;
}

//-----------------------------------------------------------------
void UpdateFreq1(long frequency_TX)
{
  if (LastFreqWriteTime != 0)
  {
    if ((millis() - LastFreqWriteTime) < 100) return;
  }
  LastFreqWriteTime = millis();
  if(frequency_TX == frequency_old_TX) return;
  program_freq1( frequency_TX );
  frequency_old_TX = frequency_TX;
}
//-----------------------------------------------------------------
```

```
//#####################################################################
//--------------------  TX Routine  -------------------------------
void TX_routine()

{     //------------------  SSB Portion  --------------------------

  PTT_SSB_Key = digitalRead( PTT_SSB );           // check to see if PTT is pressed,
  if    ( PTT_SSB_Key == LOW  )                   // if pressed do the following
  {
    do
    {
      TX_Frequency = frequency;
      frequency_tune  = TX_Frequency;             // RitFreqOffset is from Rit_Read();
      digitalWrite ( FREQ_REGISTER_BIT,   HIGH);  //
      UpdateFreq1(frequency_tune);

      Splash_MODE();
      splash_TX_freq();

      old_b = b;                                  // save original b into old_b
      b = 0x00;                                   // b is now set to wide filter setting
      Select_Multi_BW_Ored();                     // b is sent to port expander ic

      digitalWrite ( MIC_LINE_MUTE, HIGH);        // turns Q35, Q16 off, unmutes mic/line
      digitalWrite (SSB_CW, HIGH);                // this causes the ALC line to connect
      digitalWrite(TX_OUT, HIGH);                 // Truns on Q199 (pwr cntrl)(switched lo/dds)
                                                  // mutes audio to lm386
      PTT_SSB_Key = digitalRead(PTT_SSB);         // check to see if PTT is pressed
    }
    while (PTT_SSB_Key == LOW);

    b = old_b;                                    // original b is now restored
    Select_Multi_BW_Ored();                       // original b is sent to port expander

    digitalWrite(TX_OUT, LOW);                    // turn off TX stuff
    digitalWrite ( FREQ_REGISTER_BIT,   LOW);     // added 6/23/14
    digitalWrite ( MIC_LINE_MUTE, LOW);           // turns Q36, Q16 on, mutes mic/line
  } // End of SSB TX routine
```

```
 //---------------  CW Portion  -------------------------------
  TX_key = digitalRead(TX_Dit);                  //  Maybe put these on an interrupt!
  if ( TX_key == LOW)                            // was high
  {
    do
    {

      TX_Frequency = frequency;
      frequency_tune  = TX_Frequency;            // RitFreqOffset is from Rit_Read();
      digitalWrite ( FREQ_REGISTER_BIT,   HIGH); //
      UpdateFreq1(frequency_tune);

      Splash_MODE();
      splash_TX_freq();

      old_b = b;
      b = 0x00;                                  // b is now set to wide filter setting
      Select_Multi_BW_Ored();                    // b is sent to port expander ic


      digitalWrite(TX_OUT, HIGH);                // turns tx circuit on
      digitalWrite (SSB_CW, LOW);                // enables the cw pull down circuit
      digitalWrite(Side_Tone, HIGH);             // enables side-tone source
      TX_key = digitalRead(TX_Dit);              // reads dit key line
    }
    while (TX_key == LOW);                       // key still down
    b = old_b;                                   // original b is now restored
    Select_Multi_BW_Ored();
    for (int i=0; i <= 10e2; i++);               // delay
    digitalWrite(TX_OUT, LOW);                   // trun off TX cw key is now high
    digitalWrite ( FREQ_REGISTER_BIT,   LOW);    // return to DDS register 0  not in other

    digitalWrite(Side_Tone, LOW);                // side-tone off
  }
} // end  TX_routine()

//---------------------------------------------------------------------
```

```
//-------------------------- Encoder Routine --------------------

void Encoder()
{
  n = digitalRead(encoder0PinA);
  if ( encoder0PinALast != n)
  {
    if ((encoder0PinALast == LOW) && (n == HIGH))
    {
      if (digitalRead(encoder0PinB) == LOW)      //  Frequency_down
      {      //encoder0Pos--;
        frequency = frequency - Frequency_Step;
        Step_Flash();
        if ( bsm == 1 ) {
          Band_20_Limit();
        }
        else if ( bsm == 0 ) {
          Band_40_Limit();
        }
      }
      else                                       //  Frequency_up
      { //encoder0Pos++;
        frequency = frequency + Frequency_Step;
        Step_Flash();
        if ( bsm == 1 ) {
          Band_20_Limit();
        }
        else if ( bsm == 0 ) {
          Band_40_Limit();
        }
      }
    }
    encoder0PinALast = n;
  }
}

//--------------------------------------------------------
```

```
//-------------------------------------------------------
void Change_Band()
{
  if ( bsm == 1 )                          //  select 40 or 20 meters 1 for 20 0 for 40
  {
    digitalWrite(Band_Select, HIGH);
    Band_Set_40_20M();
  }
  else
  {
    digitalWrite(Band_Select, LOW);
    Band_Set_40_20M();
    IF *= -1;                              //  HI side injection
  }
}

//------------------ Band Select ----------------------------------
void Band_Set_40_20M()
{
  if ( old_bsm != bsm)                     //  this helps 5/13/14
  {
    if ( bsm == 1 )                        //  select 40 or 20 meters 1 for 20 0 for 40
    {
      frequency_default = meter_20;
      Splash_Band();
    }
    else
    {
      frequency_default = meter_40;
      Splash_Band();
      IF *= -1;                            //  HI side injection
    }
    Default_frequency();
  }
  old_bsm = bsm;                           //  this helps 5/13/14
}

//--------------------Default Frequency-----------------------------
void Default_frequency()
{
  frequency = frequency_default;
  UpdateFreq(frequency);
  splash_RX_freq();
} //  end   Default_frequency

//-------------------------------------------------------
```

```
//-----------------------------------------------------
void  Band_40_Limit()
{
  if ( frequency >= 16.3e6 )
  {
    frequency = 16.3e6;
    stop_led_on();
  }
  else if ( frequency <= 16.0e6 )
  {
    frequency = 16.0e6;
    stop_led_on();
  }
  else {
    stop_led_off();
  }
}
//-----------------------------------------------------
void  Band_20_Limit()
{
  if ( frequency >= 5.35e6 )
  {
    frequency = 5.35e6;
    stop_led_on();
  }
  else if ( frequency <= 5.0e6 )
  {
    frequency = 5.0e6;
    stop_led_on();
  }
  else {
    stop_led_off();
  }
}

//-----------------------------------------------------
void Step_Flash()
{
  stop_led_on();
  for (int i=0; i <= 25e3; i++);   // short delay
  stop_led_off();
}

//-----------------------------------------------------
void stop_led_on()                //  band edge and flash
{
  digitalWrite(Band_End_Flash_led, HIGH);
}

//-----------------------------------------------------
void stop_led_off()
{
  digitalWrite(Band_End_Flash_led, LOW);
}

//=====================================================================
```

```
//=====================================================================
void Multi_Function()                     //  pushbutton for BW, Step, Other
{
  // look into a skip rtoutine for this
  Step_Multi_Function_Button = digitalRead(Multi_Function_Button);
  if (Step_Multi_Function_Button == HIGH)
  {
    while( digitalRead(Multi_Function_Button) == HIGH ){
    }                                     // added for testing
    for (int i=0; i <= 150e3; i++);       // short delay

    Step_Multi_Function_Button1 = Step_Multi_Function_Button1++;
    if (Step_Multi_Function_Button1 > 2 )
    {
      Step_Multi_Function_Button1 = 0;
    }
  }
  Step_Function();
} // end Multi_Function()
//=====================================================================
//=====================================================================
void Step_Function()
{
  switch ( Step_Multi_Function_Button1 )
  {
  case 0:
    m = 0x08;                             // GPA3(24) Controls Function Green led
    Select_Multi_BW_Ored();
    Step_Select_Button1 = Selected_BW;    //
    Step_Select();                        //
    Selection();
    for (int i=0; i <= 255; i++);         // short delay
    break;   //

  case 1:
    m = 0x10;                             // GPA4(25) Controls Function Yellow led
    Select_Multi_BW_Ored();
    Step_Select_Button1 = Selected_Step;  //
    Step_Select();                        //
    Selection();
    for (int i=0; i <= 255; i++);         // short delay
    break;   //

  case 2:
    m = 0x20;                             // GPA5(26) Controls Function Red led
    Select_Multi_BW_Ored();
    Step_Select_Button1 = Selected_Other; //
    Step_Select();                        //
    Selection();
    for (int i=0; i <= 255; i++);         // short delay
    break;   //
  }
} // end Step_Function()
```

```
//=====================================================================
void  Selection()
{
  Step_Select_Button = digitalRead(Select_Button);
  if (Step_Select_Button == HIGH)
  {
    while( digitalRead(Select_Button) == HIGH ){
    }                                          // added for testing
    for (int i=0; i <= 150e3; i++);            // short delay

    Step_Select_Button1 = Step_Select_Button1++;
    if (Step_Select_Button1 > 2 )
    {
      Step_Select_Button1 = 0;
    }
  }
  Step_Select();
}  // end Selection()
```

```
//---------------------------------------------------------------------
void Step_Select()
{
  switch ( Step_Select_Button1 )
  {
  case 0:                          //   Select_Green
    s = 0x01;                      // GPA0(21) Controls Selection Green led
    if (Step_Multi_Function_Button1 == 0)
    {
      b = 0x00;                    // Hardware control of I.F. filter shape
      Selected_BW = Wide_BW;       // GPA7(28)LOW_GPA6(27)LOW wide
    }
    else if (Step_Multi_Function_Button1 == 1)
    {
      Frequency_Step = 100;        //  Can change this whatever step size one wants
      Selected_Step = Step_100_Hz;
    }
    else if (Step_Multi_Function_Button1 == 2)
    {
      bsm = 0;
      Change_Band();
      Encoder();
      Selected_Other = Other_1_user;
      // Other_1();
    }
    for (int i=0; i <= 255; i++); // short delay
    break;

  case 1:                          //   Select_Yellow
    s = 0x02;                      //  GPA1(22) Controls Selection Green led
    if (Step_Multi_Function_Button1 == 0)
    {
      b = 0x40;                    // Hardware control of I.F. filter shape
      Selected_BW = Medium_BW;     //  GPA7(28)LOW_GPA6(27)HIGH medium
    }
    else if (Step_Multi_Function_Button1 == 1)
    {
      Frequency_Step = 1e3;        //  Can change this whatever step size one wants
      Selected_Step = Step_1000_hz;
    }
    else if (Step_Multi_Function_Button1 == 2)
    {
      bsm = 1;
      Change_Band();
      Encoder();
      Selected_Other = Other_2_user;

      // Other_2();
    }
    for (int i=0; i <= 255; i++); // short delay
    break;
```

```
  case 2:                          //   Select_Red
    s = 0x04;                      //  GPA2(23) Controls Selection Green led
    if (Step_Multi_Function_Button1 == 0)
    {
      b = 0x80;                    // Hardware control of I.F. filter shape
      Selected_BW = Narrow_BW;     //  GPA7(28)HIGH_GPA6(27)LOW narrow
    }
    else if (Step_Multi_Function_Button1 == 1)
    {
      Frequency_Step = 10e3;       //  Can change this whatever step size one wants
      Selected_Step = Step_10000_hz;
    }
    else if (Step_Multi_Function_Button1 == 2)
    {
      Selected_Other = Other_3_user;

      // Other_3();
    }
    for (int i=0; i <= 255; i++); // short delay
    break;
  }
  Select_Multi_BW_Ored();
  Splash_Step_Size();
  Splash_BW();
}  // end Step_Select()
```

```
//--------------------------------------------------------------------------------
void Select_Multi_BW_Ored()
{
  t = s | m | b ;                // or'ed bits

  Wire.beginTransmission(0x20);
  Wire.send(0x12);               // GPIOA
  Wire.send(t);                  // port A  result of s, m, b
  Wire.endTransmission();

} // end  Select_Multi_BW_Ored()
```

```
//------------------------------------------------------------------------------
// ***************  Dont bother the code below  ****************************
// \/  \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/ \/
//------------------------------------------------------------------------------
void program_freq0(long frequency)
{
  AD9834_reset_high();
  int flow,fhigh;
  fcalc = frequency*(268.435456e6 / Reference );    // 2^28 =
  flow = fcalc&0x3fff;                              //  49.99975mhz
  fhigh = (fcalc>>14)&0x3fff;
  digitalWrite(FSYNC_BIT, LOW);                     //
  clock_data_to_ad9834(flow|AD9834_FREQ0_REGISTER_SELECT_BIT);
  clock_data_to_ad9834(fhigh|AD9834_FREQ0_REGISTER_SELECT_BIT);
  digitalWrite(FSYNC_BIT, HIGH);
  AD9834_reset_low();
} // end   program_freq0

//------------------------------------------------------------------------------

void program_freq1(long frequency_TX)
{
  AD9834_reset_high();
  int flow,fhigh;
  fcalc = frequency*(268.435456e6 / Reference );    // 2^28 =
  flow = fcalc&0x3fff;                              //  use for 49.99975mhz
  fhigh = (fcalc>>14)&0x3fff;
  digitalWrite(FSYNC_BIT, LOW);
  clock_data_to_ad9834(flow|AD9834_FREQ1_REGISTER_SELECT_BIT);
  clock_data_to_ad9834(fhigh|AD9834_FREQ1_REGISTER_SELECT_BIT);
  digitalWrite(FSYNC_BIT, HIGH);
  AD9834_reset_low();
}

//------------------------------------------------------------------------------
void clock_data_to_ad9834(unsigned int data_word)
{
  char bcount;
  unsigned int iData;
  iData=data_word;
  digitalWrite(SCLK_BIT, HIGH);                         //portb.SCLK_BIT = 1;
  // make sure clock high - only chnage data when high
  for(bcount=0;bcount<16;bcount++)
  {
    if((iData & 0x8000)) digitalWrite(SDATA_BIT, HIGH);  //portb.SDATA_BIT = 1;
    // test and set data bits
    else  digitalWrite(SDATA_BIT, LOW);
    digitalWrite(SCLK_BIT, LOW);
    digitalWrite(SCLK_BIT, HIGH);
    // set clock high - only change data when high
    iData = iData<<1;                                   // shift the word 1 bit to the left
  } // end for
} // end  clock_data_to_ad9834
```

```
//-------------------------------------------------------------------------
void AD9834_init()              // set up registers
{
  AD9834_reset_high();
  digitalWrite(FSYNC_BIT, LOW);
  clock_data_to_ad9834(0x2300);  // Reset goes high to 0 the registers
                                 // and enable the output to mid scale.
  clock_data_to_ad9834((FREQ0_INIT_VALUE&0x3fff)|AD9834_FREQ0_REGISTER_SELECT_BIT);
  clock_data_to_ad9834(((FREQ0_INIT_VALUE>>14)&0x3fff)|AD9834_FREQ0_REGISTER_SELECT_BIT);
  clock_data_to_ad9834(0x2200);  // reset goes low to enable the output.
  AD9834_reset_low();
  digitalWrite(FSYNC_BIT, HIGH);
} //  end   AD9834_init()

//-------------------------------------------------------------------------
void AD9834_reset()
{
  digitalWrite(RESET_BIT, HIGH);  // hardware connection
  for (int i=0; i <= 2048; i++);  // small delay

  digitalWrite(RESET_BIT, LOW);   // hardware connection
}  // end AD9834_reset()

//-------------------------------------------------------------------------
void AD9834_reset_low()
{
  digitalWrite(RESET_BIT, LOW);
} // end AD9834_reset_low()

//.........................................................................
void AD9834_reset_high()
{
  digitalWrite(RESET_BIT, HIGH);
} // end  AD9834_reset_high()

//^^^^^^^^^^^^^^^^^^^^^^^^^  DON'T BOTHER CODE ABOVE  ^^^^^^^^^^^^^^^^^^^^^^^^^
//=========================================================================
```

```
//------------------------Display Stuff below----------------------------------

// All the code for display below seems to work well. 6-18-14

//------------------- Splash RIT -----------------------------------------------
void splash_RIT()      // not used
{
  if ( old_RitFreqOffset != RitFreqOffset)  // only if RIT changes
  {
    lcd.setCursor(16, 0);
    lcd.print(txt70);                       // spaces

    lcd.setCursor(16, 0);
    stringRIT = String( RitFreqOffset, DEC);

    lcd.print(stringRIT);
  }
  old_RitFreqOffset = RitFreqOffset;        // test for Rit change
}
```

```
//--------------------------------------------------------------------------
void splash_TX_freq()
{
  long TXD_frequency;                   // ADDED 6-18-14 OK

  if ( bsm == 1 )                       // test for 20M
  {
    TXD_frequency = frequency_tune ;
  }

  else if ( bsm == 0 )                  // test for 40M
  {
    TXD_frequency = frequency_tune ;
  }
  //-------------------------------------------------

  if ( TXD_frequency < 5.36e6 )
  {
    TXD_frequency = TXD_frequency + 9e6;
  }

  else if ( TXD_frequency > 15.95e6 )
  {
    TXD_frequency = TXD_frequency - 9e6;
  }
  //-------------------------------------------------

  lcd.setCursor(3, 1);
  stringFREQ = String(TXD_frequency / 10, DEC);
  lcd.print(stringFREQ);

}
```

```
//--------------------------------------------------------------------------
void splash_RX_freq()
{
  long RXD_frequency;                    // ADDED 6-18-14 OK

  if ( old_frequency_tune != frequency_tune )
  {
    if ( bsm == 1 )                      // test for 20M
    {
      RXD_frequency = frequency_tune ;
    }

    else if ( bsm == 0 )                 // test for 40M
    {
      RXD_frequency = frequency_tune ;
    }
    //-----------------------------------------

    if ( RXD_frequency < 5.36e6 )
    {
      RXD_frequency = RXD_frequency + 9e6;
    }

    else if ( RXD_frequency > 15.95e6 )
    {
      RXD_frequency = RXD_frequency - 9e6;
    }
    //-----------------------------------------

    lcd.setCursor(3, 0);
    lcd.print(txt72);                    // spaces

    lcd.setCursor(3, 0);
    stringFREQ = String(RXD_frequency , DEC);
    lcd.print(stringFREQ);
  }
  old_frequency_tune = frequency_tune;
}
```

```
//----------------------------------------------------------------
void Splash_Band()
{
  if ( bsm == 1 )            // test for 20M
  {
    lcd.setCursor(17, 1);
    lcd.print(txt66);        // 20 meters
  }
  else
  {
    lcd.setCursor(17, 1);
    lcd.print(txt67);        // 40 meters
  }
}
```

```
//--------------------------------------------------------------------------------

void Splash_Step_Size()
{
  if ( old_Frequency_Step != Frequency_Step ) //
  {
    lcd.setCursor(5, 2);
    lcd.print(txt71 );                        // spaces

    lcd.setCursor(5, 2);

    string_Frequency_Step = String(Frequency_Step, DEC);
    lcd.print(string_Frequency_Step);
  }
  old_Frequency_Step = Frequency_Step;        // test for Rit change
}

//--------------------------------------------------------------------
```

```
//-----------------------------------------------------------------
void Splash_BW()
{
  if ( old_b != b )
  {
    if ( b == 0x00 )
    {
      lcd.setCursor(3, 3);
      lcd.print(txt170);
      lcd.setCursor(3, 3);
      lcd.print(txt140);  // wide
    }
    else if ( b == 0x40 )
    {
      lcd.setCursor(3, 3);
      lcd.print(txt170);
      lcd.setCursor(3, 3);
      lcd.print(txt150);  // medium
    }
    else {
      lcd.setCursor(3, 3);
      lcd.print(txt170);
      lcd.setCursor(3, 3);
      lcd.print(txt160);  // narrow
    }
  }
  old_b = b ;
}
```

```
//--------------------------------------------------------------------
void Splash_MODE()
{
  if ( old_PTT_SSB_Key != PTT_SSB_Key )
  {
    if ( PTT_SSB_Key == LOW )
    {
      lcd.setCursor(17, 3);
      lcd.print(txt69);
      lcd.setCursor(17, 3);
      lcd.print(txt135);    // SSB
    }
    else
    {
      lcd.setCursor(17, 3);
      lcd.print(txt69);
      lcd.setCursor(17, 3);
      lcd.print(txt132);    // CW
    }
  }
  old_PTT_SSB_Key = PTT_SSB_Key;
}
//--------------------------------------------------------------------------
//--------------------------------------------------------------------------
//stuff above is for testing using the Display Comment out if not needed
//--------------------------------------------------------------------------
```

```
uint32_t TimerOverFlow(uint32_t currentTime)
{
  return (currentTime + CORE_TICK_RATE*(1));     //the Core Tick Rate is 1ms
}

//------------------ Debug data output ------------------------------

void    serialDump()
{
  loopStartTime   = millis();
  loopsPerSecond  = loopCount - lastLoopCount;
  loopSpeed       = (float)1e6 / loopsPerSecond;
  lastLoopCount   = loopCount;

  Serial.print    ( "uptime: " );
  Serial.print    ( ++printCount );
  Serial.println  ( " seconds" );

  Serial.print    ( "loops per second:    " );
  Serial.println  ( loopsPerSecond );
  Serial.print    ( "loop execution time: " );
  Serial.print    ( loopSpeed, 3 );
  Serial.println  ( " uS" );

  Serial.print    ( "Freq Rx: " );

  Serial.println  ( frequency_tune - IF );
  Serial.println  ( RX_frequency );

  Serial.print    ( "Freq Tx: " );

  Serial.println  ( frequency - IF );
  Serial.println  ( TX_frequency );

  Serial.print    ( "RIT: " );
  Serial.println  (  RitFreqOffset );

  Serial.print    ( "BW: " );
  Serial.println  ( );
  //   Serial.println  (  RitFreqOffset );

  Serial.print    ( "BAND: " );
  Serial.println  ( );
  //    Serial.println  (  RitFreqOffset );

  Serial.print    ( "MODE: " );
  Serial.println  ( );
  //    Serial.println  (  RitFreqOffset );

  Serial.print    ( "STEP: " );
  Serial.println  (  Frequency_Step );

  Serial.println  ();

} // end serialDump()
```