ext60

```
/*
 BITX program for 60 M (USA allocation) with added channels and CW
 Three tuning modes
 V 1.0.9   Don Cantrell, ND6T 24 October 2017
 Compiles under etherkit Si535 library v 2.0.1
 Runs on modified BITX-40 Raduino hardware
 This source file is under General Public License version 3.0
 S meter to A1,  Reverse power to A2, Forward power to A3, Keying speed to A6
 */
#include <si5351.h>
Si5351 si5351;
#include <LiquidCrystal.h>
LiquidCrystal lcd(8,9,10,11,12,13);

//****declare variables****

  float sm = 0;           //"S" meter value
  int offset;             //CW offset ( fixed in U.S)
  int sidetone=700;       //Sidetone frequency
  int wpm = 20;
  float F;                //Forward RF output (PEP watts RMS)
  float FP;               //SSB ""
  float R;                //Reverse RF output (PEP watts RMS)
  float RP;               //SSB ""
  int p ;                 //Timing period (milliseconds) for keyer function
  int channel = 1;        //Channel number
  long tune;              //Tuning knob position
  long oldTune;           //Previous tune value
  long count = 0;         //Timeout counter
  unsigned long post;     //Timing milepost
  unsigned long post1;    //Timing milepost 1
  long BFO = 11999038;    //My BFO frequency (11999038)
  long LO = BFO + 5330500;//Local Oscillator for Upper sideband,CH.1
  long frequency;

void setup() {

    lcd.begin(16, 2);
    si5351.init(SI5351_CRYSTAL_LOAD_8PF,25004920,0); //My actual ref osc freq.
    si5351.set_pll(SI5351_PLL_FIXED, SI5351_PLLA);
    si5351.set_freq(LO * 100, SI5351_CLK2);          //Program the synthesizer

    pinMode(4, INPUT_PULLUP); //Dash input on Plug 3 pin 4
    pinMode(5, INPUT_PULLUP); // Dot input on Plug 3 pin 3
    pinMode(6, OUTPUT);       //Sidetone from Plug 3 pin 2
    pinMode(7, OUTPUT);       //T/R keying for CW Plug 3 pin 1

//// Splash  ///////
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("ext60 v1.0.9");
  delay(3000);
}
```

```
void loop() {

   pwrTest();           //Measure power and VSWR

   cwTest();            //Look for key closure

   sm=analogRead(A1);   //Read S meter value

   tune = analogRead(A7);//Read the tuning input on analog pin 7:

//Set switching at knob limits and increment channel selection
    if (tune>1000){
          ++channel;
          post1=millis(); //Stake a time post for channel update limit
          }
    if (channel > 6)channel = 1;
    if (tune < 20)channel = 0;

 switch (channel) {
    case 0:
      ShuttleTuning();
        break;
    case 1:
      frequency = 5330500;
      offset = 1500;
         break;
    case 2:
      frequency = 5346500;
      offset = 1500;
         break;
      case 3:
      frequency = 5357000;
      offset = 1500;
         break;
      case 4:
      frequency = 5371500;
      offset = 1500;
          break;
      case 5:
      frequency = 5403500;
      offset = 1500;
         break;
      case 6:
        VCO();
        offset = sidetone;
         break;
     }
  if(millis()-post1<100){     // Update 5351 only if under 100 ms
  LO = BFO + frequency;
  program();
  show();
    }
  if(millis()-post>2e3){  //Refresh display every 3 seconds
    show();    //(Adds 2 seconds to the delay() below)
    F=0;       //Reset power reading
    R=0;
    post=millis();        // Reset that timer
    }
 if (tune>1000)delay(1000); // Slow channel selection to make it easier
}

//****Functions****

void program(){
  si5351.set_freq(LO * 100, SI5351_CLK2);  //Program the synthesizer
}

void show() {  //Display function

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("#");
  lcd.print(channel);
  lcd.print("=");
  lcd.print ((LO-BFO)/1e6,6);//Calculate & show frequency
   if(tune>560)lcd.print(" >");
   if(tune<464)lcd.print(" <");
   if (tune>464 && tune<560)lcd.print(" I"); //Idle indicator
  lcd.setCursor(14,0);
  lcd.print((analogRead(A6)/30));//Display keying speed
    if(analogRead(A6)<300){     //or Straight Key mode
      lcd.setCursor(14,0);
      lcd.print("SK");}
  lcd.setCursor(0,1);
  if(F>=1){  //If RF power present replace S meter with power.
   lcd.print(F,0);
   lcd.print("W  SWR=");
   lcd.print((1+sqrt(R/F))/(1-sqrt(R/F)),1);
   lcd.print(":1");
   }
  else{  //Otherwise display S meter
    if (sm>=110){lcd.print("S9+20");}
    if (sm>=80&&sm<110){lcd.print("S9+10");}
    if (sm>=45&&sm<80){lcd.print("S9");}
    if (sm<45){lcd.print("S");
    lcd.print(sm/5,0);}
    lcd.setCursor(0,1);
    }
}

 void ShuttleTuning() {
 while(tune<1000) {
     tune = analogRead(A7);// Read the input on analog pin 7

    if (tune>560)up();   //Establish tuning direction
    if (tune<464)down();
    if(millis()-post1<1000)show();//Display then freeze
    delay(500);// Slow to ease tuning
    }
 }

void up() {
    LO = LO + round (pow((tune - 560)/5,3)/100);     //Increase local osc frequency
   program();
    post1=millis();
}
```

```
void down() {
    {LO = LO - round (pow((464 - tune)/5,3)/100);}        //Decrease local osc
frequency
    program();
    post1=millis();
}

 void VCO(){  //Voltage controlled oscillator
  while (analogRead(A7)<1000){
  pwrTest();
  cwTest();
  sm = analogRead(A1);// Read signal level on A2
  tune=0;
  for(int i = 0; i<100; i++)tune=tune+analogRead(A7);//Oversammple 100X
  tune=tune/6.66;              //Scale to available range
  if(tune/10!=oldTune){       //Update only when changed
    frequency=((tune)+5351500);
    LO=BFO+frequency;
      program();
      show();
      delay(100);
      }
  oldTune=tune/10;
  if(millis()-post>2e3){  //Refresh display every 3 seconds
    show();    //
    F=0;       //Reset power reading
    R=0;
    post=millis();        // Reset that timer
    }
  }
  if(analogRead(A7)<1010)warning();
}

void warning(){ //Tuned to high end. Give operator a chance to return
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Warning!! High!");
  lcd.setCursor(0,1);
  lcd.print("Return to scan?");
  delay(5000); // Wait 5 seconds to be sure
  if(analogRead(A7)<1000)VCO; //If tuned lower, return to VCO
}

void cwTest() { //Look for key closure
  if(((digitalRead(4)==LOW)&&(analogRead(A6)>=300)) || (digitalRead(5)==LOW))
{CW();}//Is the key active?

  digitalWrite(7, LOW); // Restore T/R switching from CW mode
}

void pwrTest() {
  FP=analogRead(A3)/(3e4/analogRead(A3)+1); //Read Forward RF power
   if(FP>F)F=FP;
   RP=analogRead(A2)/(3e4/analogRead(A2)+1); //Read Reverse RF power
   if(RP>R)R=RP;
}
```

```
void CW() {             //CW modes
  digitalWrite(7,HIGH); // Key T/R relays and do the setup while they activate
  wpm = analogRead(A6)/30; //Read CW speed pot and set WPM rate
  p = 1200/wpm;   // convert speed to milliseconds

  if (wpm < 10)sk();// Read speed control to switch to Straight Key mode
  if (wpm < 10)return;

  //Iambic keyer
  while (count < 1e5) { // Delay time after last action to return to normal SSB
    if(digitalRead(4)==LOW)dah();
    if(digitalRead(5)==LOW)dit();
    count++;} //Increment time-out for CW routine
      count=0; // Reset the CW timeout
}

void dit() {
   si5351.set_freq((frequency+offset) * 100 , SI5351_CLK1); //Key on CW transmit
frequency
    tone(6,sidetone);
    delay(p);
    noTone(6);
  si5351.output_enable(SI5351_CLK1, 0);    // Unkey transmit
    delay(p);
  count=0;
}

void dah() {
   si5351.set_freq((frequency+offset) * 100 , SI5351_CLK1); //Key on CW transmit
frequency
    tone(6,sidetone);
    delay(3*p);
    noTone(6);
     si5351.output_enable(SI5351_CLK1, 0); // Unkey transmit
    delay(p);
   count=0;
}

void sk() {  //Straight Key mode
  while (count < 2000) { // Delay time after last action to return to normal SSB
   if(digitalRead(5)==LOW)post=millis();     //Set post for display timing
     while(digitalRead(5)==LOW){               //Key down
      si5351.set_freq((frequency+offset) * 100 , SI5351_CLK1);
      tone(6,sidetone); //Sidetone
     if(millis()-post>500){ //If keyed for more than half second, read power
       F=analogRead(A3)/(3e4/analogRead(A3)+1); //Read Forward RF power
       R=analogRead(A2)/(3e4/analogRead(A2)+1); //Read Reverse RF power
       show();
       delay(100);
       }
    count=0; //Reset counter
  }
    {si5351.output_enable(SI5351_CLK1, 0); // Unkey transmit
     noTone(6);
    }
     count++;
  }
   count=0; // Reset the CW timeout
}
```