

along with a block diagram and specifications. You can access them through the Reference link on the right hand sidebar, these are all in pdf format. I'm still working on some things and there might be a tweak or two before I send the release board out for fabrication. The previous board design (REVA shown on the February post) was a good learning exercise and was well worth the effort, one more clean up cycle should do it. If anyone has any comments or suggestions please post them in Design Discussion are of the forum.

I have also posted the schematics and PCB layout source files in the Design Files/Tools section as well. You will need to download and install the appropriate design tools to view these files.

Thanks, Steve K1EL

Files Updated

September 28th, 2009

I finished the final tweaks to the oQ PCB and sent it out for fabrication on 9/25. I have updated the schematic, PCB data files, and schematic data files. These now match the board that is being made. I also updated the Specification as well as a making a couple of website tweaks. This site is built upon WordPress which is a very nice package but it does take lots of work to customize. Even the simplest changes or additions require reading and reading.... In any case it's pretty close to what I want now.

As soon as the board arrives, sometime around 10/6, I will start documenting the board bring up and make regular posts.

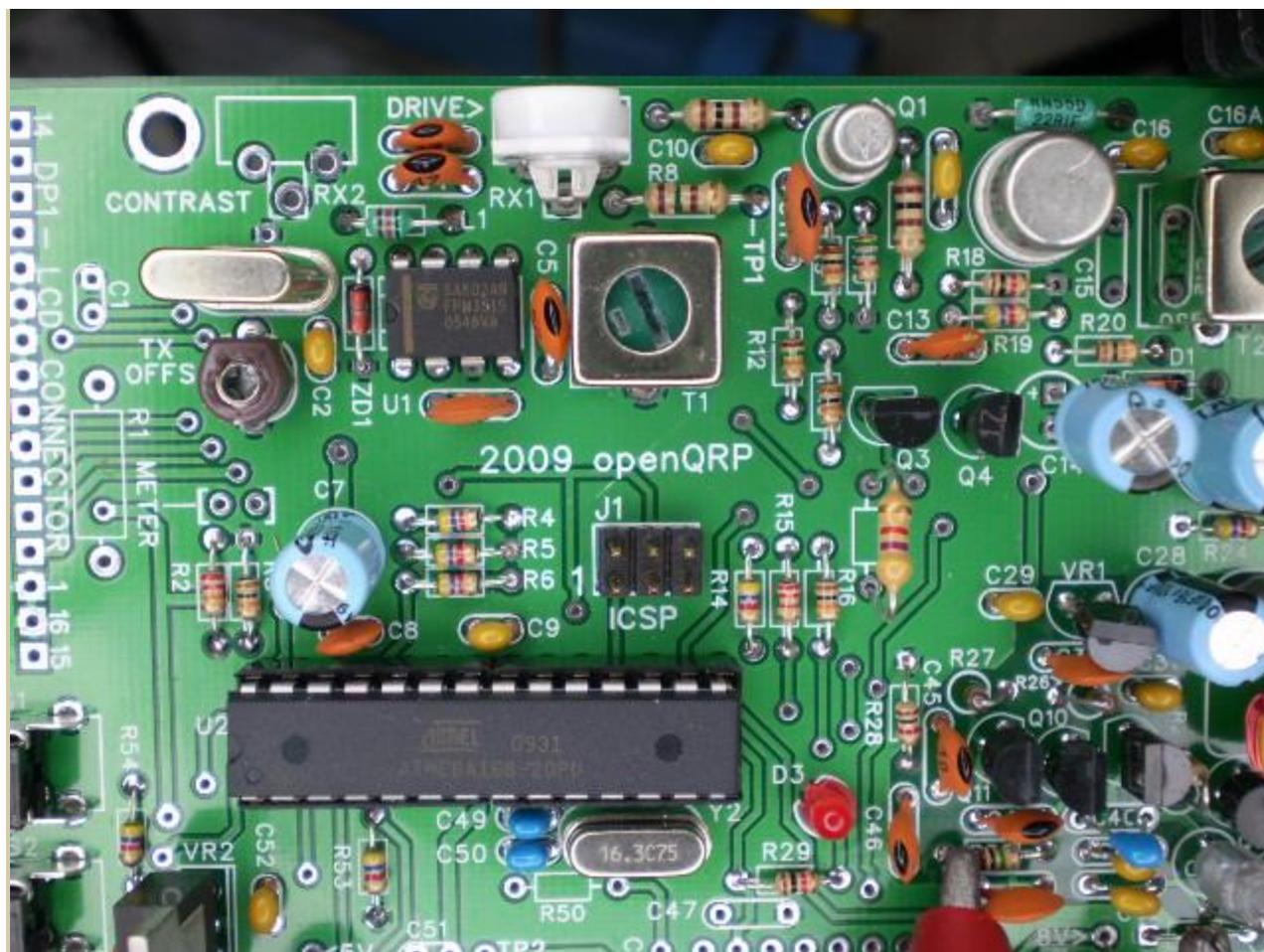
In the final version I changed the final to an IRF510 MOSFET, these are easy to find and work much better than a standard bipolar design. With the IRF510 I get a nice solid clean 8 watts out.

Transmitter Works !

October 29th, 2009

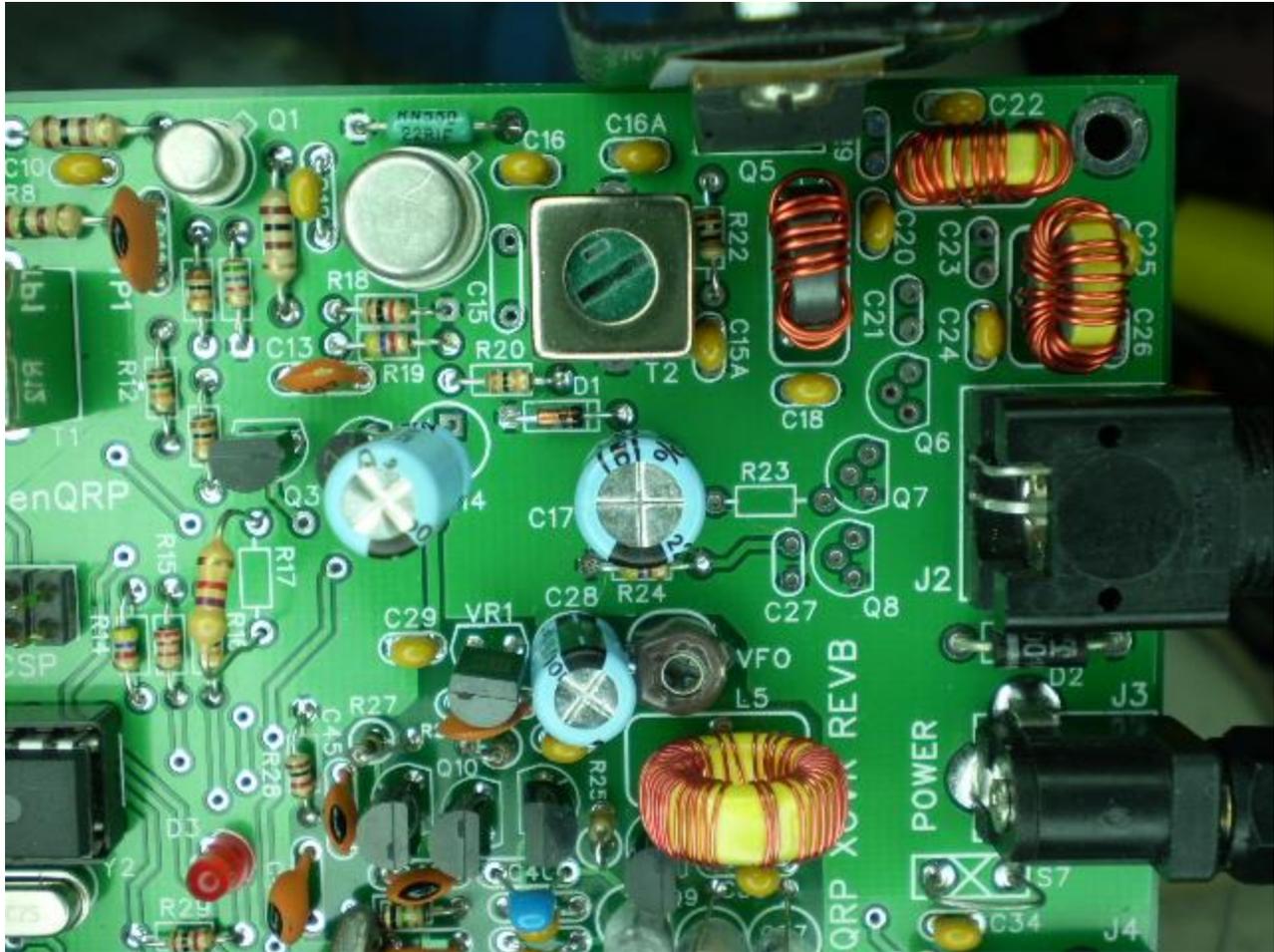
The transmitter is installed and works reliably. I'll cover some of the highlights of the design in this installment. I'm open to criticism on my calculations, I am sticking my digital neck out pretty far on some of this analog world design.

TX Mixer and First Driver Stage



A small amount of VFO feed is taken from the VFO buffer and fed into one port of a NE602A mixer. It is mixed with the local crystal oscillator (4.9152 MHz) to produce sum and difference products. The sum product ($4.9152 + 2.084 = 7.0$) is filtered by a 42IF123 IF transformer. This transformer is normally used for 10.7 MHz IF applications but in this case we pad the resonant frequency down with a parallel capacitor of 56 pf. We get a nice peak at 7 MHz by adjusting the slug in the top of the transformer. The primary side is a high impedance which allows a fairly hi Q filter with a low impedance secondary. The secondary feeds into the base circuit of Q1 which has an impedance of (10K in parallel with 5.6K) about 3.6K. The output impedance of the NE602A is about 1.5K which is connected to the center tap of the primary. The turns ratio of primary to secondary is 5T to 2T so the reflected impedance of 3.6K back to the NE602A is $(2/5)^2$ times 3.6K = 576 ohms. Not a perfect match but close enough to work ok. Q1 is the first driver amplifier that has adjustable gain via adjustable trimmer RX1. The output of this stage has a low impedance of about 100 ohms.

Second Driver Stage and Final

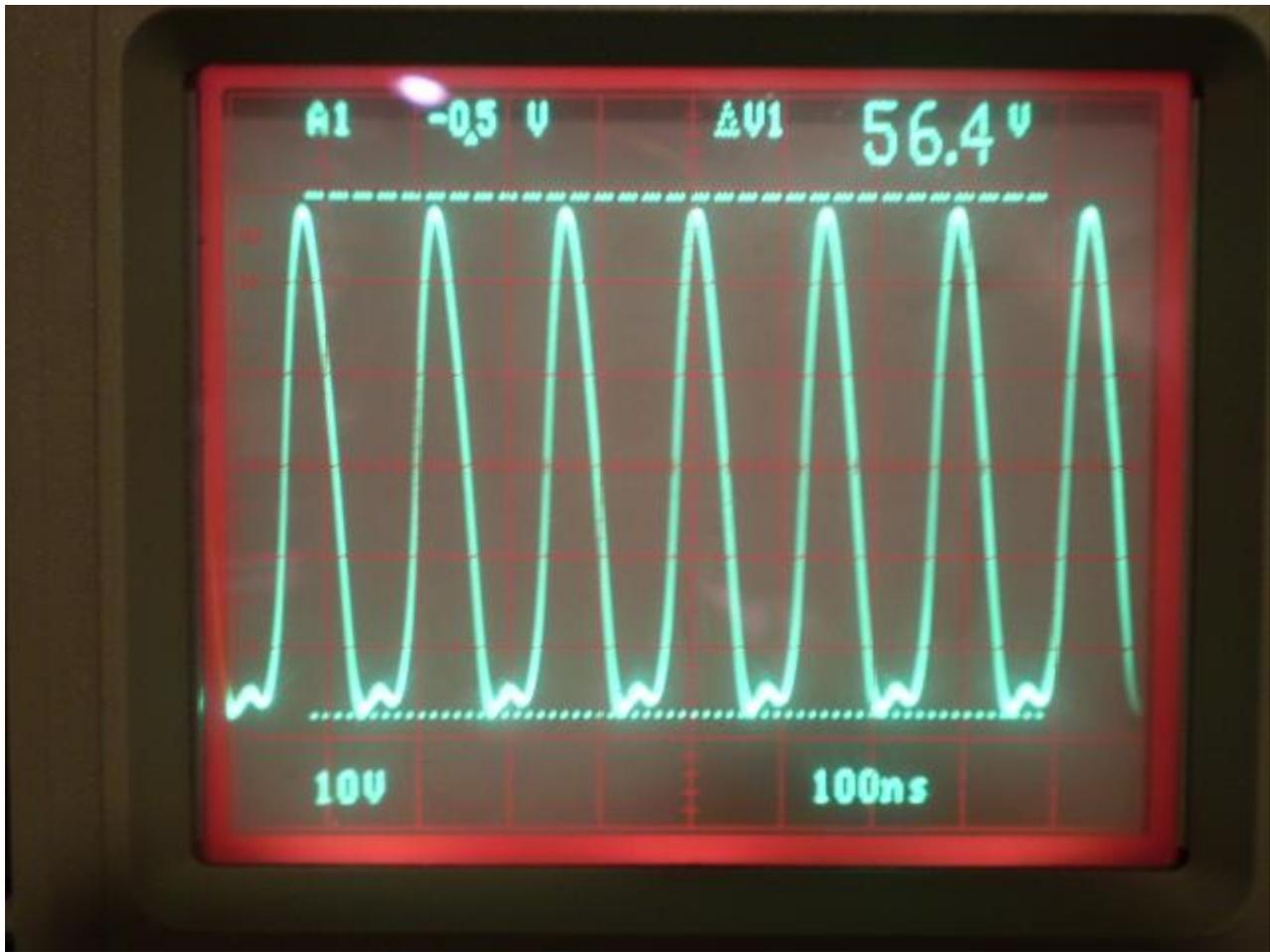


The input impedance of the second driver stage, Q2, is (1K in parallel with 4.7K) 824 ohms. The output of the first stage (100 ohms) plus the reactance of the coupling cap C12 (220 pf=100 ohms at 7MHz) is 200 ohms, a bit of a mismatch there as well. This driver stage uses a second IF transformer to drive the gate of the IRF510 MOSFET final amplifier. Originally this was set up to be tuned at 7MHz but seems to work better as an untuned broadband transformer. More investigation is due here. In any case it transforms the relatively high impedance of the collector circuit to the low impedance of the gate drive. The MOSFET gate input capacitance is quite high (around 150 pF) so a low impedance drive is required to overcome this. The MOSFET gate is biased to 5.1V by a zener diode in the gate circuit. This gives us a bias current around 40 ma. When not in transmit mode the bias is removed which drops the idle current to near zero.

Tx Test setup

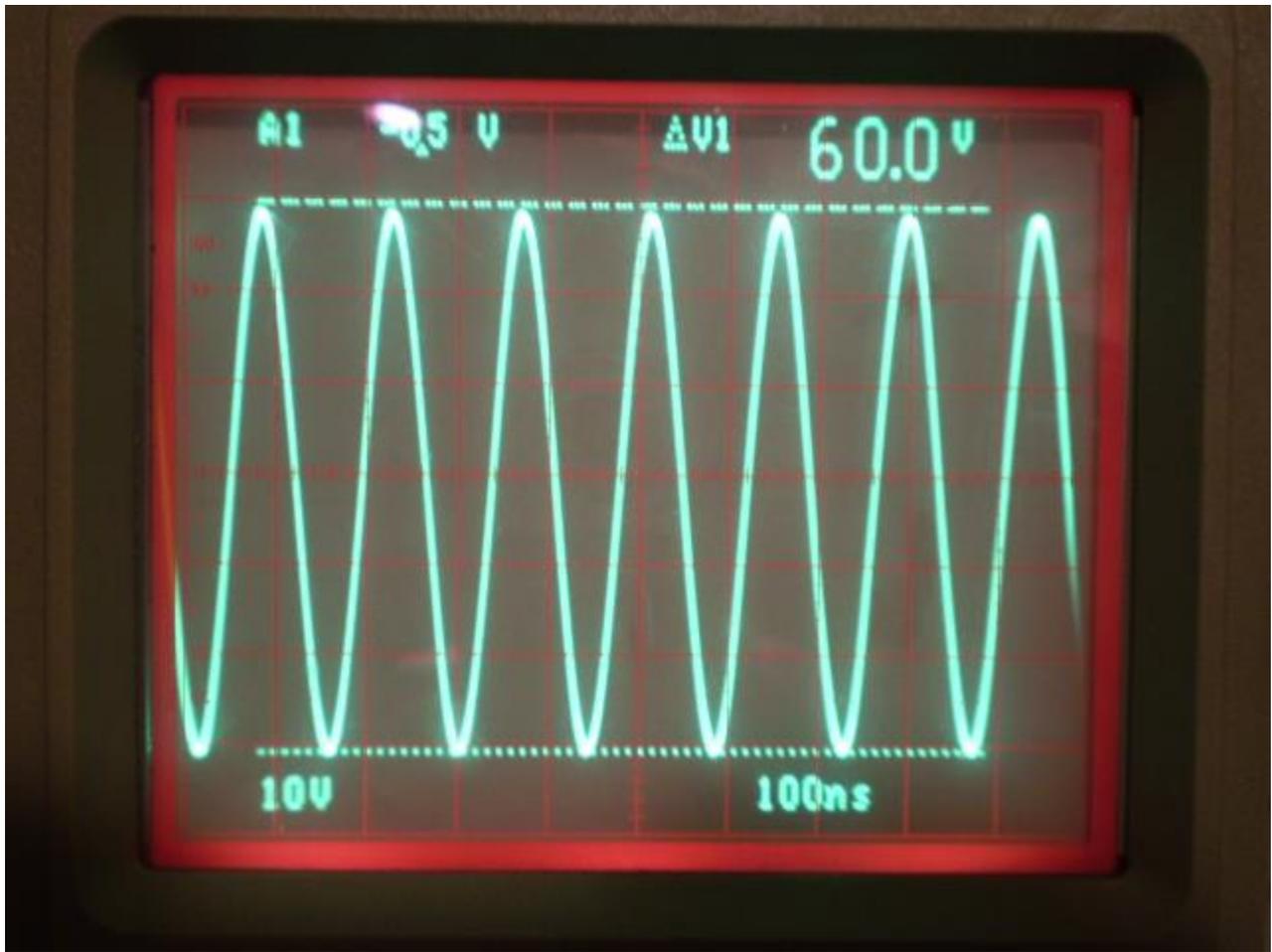


Waveform at Drain of IRF510 Final



Output Waveform: 60V Peak to Peak or $(60/2)$ times $.7071 = 21.21$ RMS Volts

Calculated Power = E^2/R Using measured values: $(21.21V)^2 / 50\Omega = 9$ watts



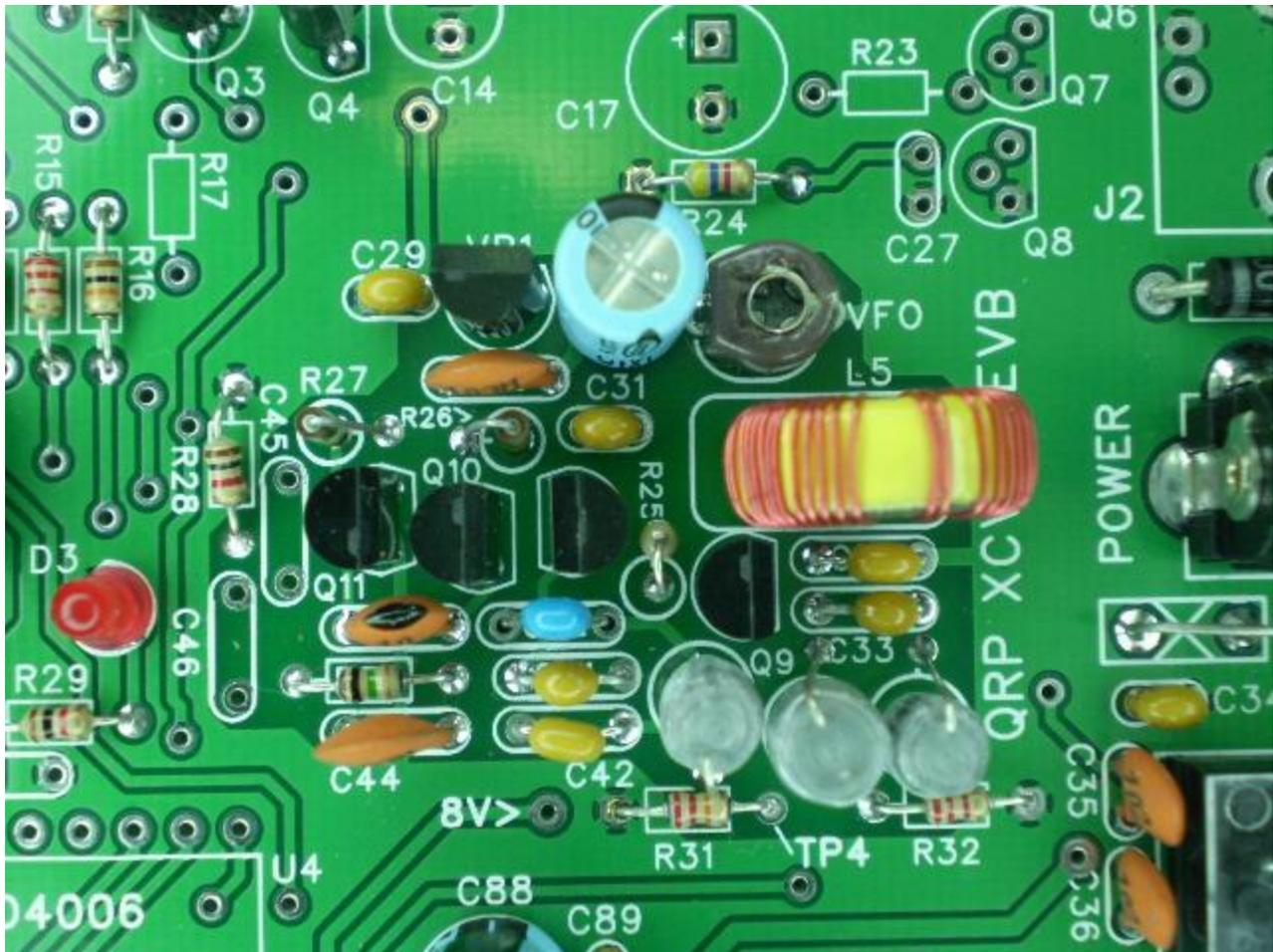
Power Meter: 8+ watts ! (meter accuracy spec'd at +/- 10%)



VFO Build and Test

October 29th, 2009

The prototype base chassis plate is on order, it'll be a couple more days until it arrives. Since the LCD display is mounted to the main PCB and is also aligned to the chassis front panel I will hold off on the LCD interface for now. Instead I decided move to the VFO section. I also want to jump ahead and get the transmitter working since I am a little uncertain about how the new IRF510 final is going to work in this PCB revision.



VFO stands for variable frequency oscillator and it determines the frequency the radio operates. I designed the VFO to run in the range from 2.0848 MHz to about 2.1648 MHz. You ask, "I thought this transceiver is supposed to cover the 40 meter CW band 7.0 to 7.080 MHz?" The VFO signal is mixed with a crystal frequency of 4.9152 MHz in the first transmit mixer. The mixing process combines the two frequencies in a nonlinear way to produce two new frequencies: (crystal+VFO) and (crystal-VFO). ($4.9152+2.0848=7\text{MHz}$ and $4.9152-2.0848=2.8304\text{MHz}$). Side-note, if the two frequencies were combined linearly (.i.e. non-distorted) the two frequencies would not be mixed but preserved as two separate frequencies. It's the non-linear transform that fosters the production of the sum and difference products, plus a whole bunch of other undesired frequencies. In our case we are only interested in the difference frequency. This gives us our coverage limits as $4.9152+2.0848=7\text{MHz}$ and $4.9152+2.1648=7.080\text{MHz}$.

The receive chain also uses a mixer to combine incoming RF and a local oscillator in a slightly different way which we cover when we get to the receiver.

The VFO uses a Colpitts topology with a split capacitance providing positive feedback to maintain oscillation. A varactor diode is used to vary the oscillation frequency. A varactor diode is a useful device in that its capacitance can be made to vary widely by applying a variable DC voltage. This VFO design is based on a standard implementation, used in many low cost CW

transceivers, most notably Dave Benson's SW series. Varactor tuned VFOs have fallen out of favor in the past couple of years due to the availability of DDS Digital Synthesis ICs from Analog Devices and other companies. I chose a varactor for two reasons, the first being that DDS ICs are only available in fine pitch surface mount packages, and two it adds a level of hardware and firmware complexity I want to avoid for this radio. But since I have the CPU to control it, I'll leave that for a future upgrade.

There are two significant disadvantages of varactor tuning, the first is the limited frequency coverage. Typical capacitance swings of 80 to 250pF are easily obtained but it's not a linear change with voltage. The second disadvantage is poor temperature stability. I designed the VFO to be as stable as possible so that I only have one contribution to drift from the varactor.

There are several things I did to improve stability. Polystyrene caps, small multi-layer ceramic caps, and a separate 8 volt regulator for the VFO help a lot. Also voiding the ground fill out of the VFO area reduces changes in capacitance due to physical dimension changes with temperature. With the varactor control grounded I see about 40 Hz drift from a cold start over a 4 hour period. With the varactor active, the frequency is less stable of course, it seems to be around 100-130 Hz from cold start but I have to wait until I get the tuning pots installed and wired nicely.

One thing I did a little differently in this design was to provide a stage of buffering between the VFO and the feeds to the Tx and Rx Mixers. There is an additional stage of buffering provided to drive the frequency counter input on the CPU.

One final note on the VFO, one thing I really like in a VFO design is a variable cap trimmer to allow for easy calibration. It cost about 45 cents and reduces temperature stability slightly but well worth it.

I will cover the transmitter build next, I'll take a couple of pictures to post tomorrow,

73 Steve K1EL

Arduino Iambic Keyer

October 23rd, 2009

I haven't had a great deal of time to work on the board this week. What little time I did have I spent on Arduino aspects of the project. I started with a small sketch that simply flashes and LED.

Today I present a basic iambic mode keyer that has adjustable speed and can be configured to operate in iambic mode a or b. At this point all it does is flash an LED in Morse, but it's a good foundation to build upon. There really isn't anything novel about it, it's the same basic algorithm used by many different keyers. Over the weeks I will add auto space, Ultimatic, and straight keying modes. I'll also include the ability to swap paddles. But this will hold us for quite a while. You can cut and paste this into an empty Arduino sketch, it's completely self-contained. It'll run on any Arduino board, just wire up a paddle and it will flash the LED. Add a pushbutton switch and you can select between iambic mode a and b. It would be very simple to

add a keying output in place of the LED drive, probably a good idea to include a 2N2222 or 2N7000 buffer stage.

```
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//
//  Iambic Morse Code Keyer Sketch
//  Copyright (c) 2009 Steven T. Elliott
//
//  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:
//
//  Free Software Foundation, Inc., 59 Temple Place, Suite 330,
//  Boston, MA 02111-1307 USA
//
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//
//                               openQRP CPU Pin Definitions
//
////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////
//
//  Arduino  OQ PCB                                OQ PCB  Arduino
//  -----  -----                                -----  -----
//
//  Dig0      RESET-----PC6 |1          28| PC5-----KEY_TX      Dig19 Ana5
//  Dig1      SER_REPLY-----PD0 |2  o   i 27| PC4-----LCD_E       Dig18 Ana4
//  Dig2      SER_CMD-----PD1 |3  o   i 26| PC3-----CMD_PB       Dig17 Ana3
//  Dig3      WIDE_SEL-----PD2 |4  o   i 25| PC2-----PB_NET      Dig16 Ana2
//  Dig4      SIDETONE-----PD3 |5  o   i 24| PC1-----ANT_XO      Dig15 Ana1
//  Dig4      LCD_D4-----PD4 |6  o   i 23| PC0-----MON_12V     Dig14 Ana0
//
//  VCC-----VCC |7  o   o 22| GND-----AGND
//  GND-----GND |8  o   o 21| AREF-----AREF
//  XTAL1-----PB6 |9  i   o 20| AVCC-----AVCC
//  XTAL2-----PB7 |10 i   o 19| PB5-----RIT_ON       Dig13
//  Dig5      FREQ_CNT-----PD5 |11 o   o 18| PB4-----NAR_SEL      Dig12
//  Dig6      LCD_D5-----PD6 |12 o   o 17| PB3-----LCD_RS       Dig11
//  Dig7      LCD_D6-----PD7 |13 o   o 16| PB2-----LPDL        Dig10
//  Dig8      LCD_D7-----PB0 |14 o   o 15| PB1-----RPDL        Dig9
//
//
//                               ATmega168
//
//  Digital Pins
//
int          widePin = 2;          // Wide filter select
```

```

int      STPin    = 3;      // Sidetone output pin
int      LPin     = 9;      // Left paddle input
int      RPin     = 10;     // Right paddle input
int      narPin   = 12;     // Narrow filter select
int      ritPin   = 13;     // RIT & LED share digital pin 13
int      ledPin   = 13;     //
int      antXOPin = 15;     // Antenna changeover control
int      cmdPin   = 17;     // Command pushbutton input
int      keyPin   = 19;     // Transmitter key output
int      lcdEPin  = 18;     // LCD E control output
//
// Analog Pins
//
int      PBnetPin = 2;      // Pushbutton network input
int      pwrPin   = 0;      // Voltage monitor input
//
//
//
// keyerControl bit definitions
//
#define    DIT_L      0x01    // Dit latch
#define    DAH_L      0x02    // Dah latch
#define    DIT_PROC   0x04    // Dit is being processed
#define    PDLSWAP    0x08    // 0 for normal, 1 for swap
#define    IAMBICB    0x10    // 0 for Iambic A, 1 for Iambic B
#define    ULTIMATIC  0x20    // 1 for ultimatic
#define    AUTOSPACE  0x40    // 1 for autospace mode
#define    STRAIGHT   0x80    // 1 for straight key mode
//
//
//
// Library Instantiations
//
//
//
//
// Global Variables
//
ULONG    ditTime;          // No. milliseconds per dit
UCHAR    keyerControl;
UCHAR    keyerState;
//
//
//
// State Machine Defines
enum KSTYPE {IDLE, CHK_DIT, CHK_DAH, KEYED_PREP, KEYED, INTER_ELEMENT };
//
//
// System Initialization
//

```

```

////////////////////////////////////
//
void setup() {
  // Setup outputs
  pinMode(ledPin, OUTPUT);      // sets the digital pin as output
  // Setup control input pins
  pinMode(LPin, INPUT);        // sets Left Paddle digital pin as input
  pinMode(RPin, INPUT);        // sets Right Paddle digital pin as input
  pinMode(cmdPin, INPUT);      // sets PB analog pin 3 as input
  digitalWrite(ledPin, LOW);   // turn the LED off
  keyerState = IDLE;
  keyerControl = 0;
  loadWPM(15);                 // Fix speed at 15 WPM
}
////////////////////////////////////
//
//
//  Main Work Loop
//
////////////////////////////////////
//
void loop()
{
  static long ktimer;
  int debounce;

  // Basic Iambic Keyer
  // keyerControl contains processing flags and keyer mode bits
  // Supports Iambic A and B
  // State machine based, uses calls to millis() for timing.

  switch (keyerState) {
    case IDLE:
      // Wait for direct or latched paddle press
      if ((digitalRead(LPin) == LOW) ||
          (digitalRead(RPin) == LOW) ||
          (keyerControl & 0x03)) {
        update_PaddleLatch();
        keyerState = CHK_DIT;
      }
      break;
    case CHK_DIT:
      // See if the dit paddle was pressed
      if (keyerControl & DIT_L) {
        keyerControl |= DIT_PROC;
        ktimer = ditTime;
        keyerState = KEYED_PREP;
      }
      else {
        keyerState = CHK_DAH;
      }
      break;

    case CHK_DAH:
      // See if dah paddle was pressed
      if (keyerControl & DAH_L) {

```

```

        ktimer = ditTime*3;
        keyerState = KEYED_PREP;
    }
    else {
        keyerState = IDLE;
    }
    break;
case KEYED_PREP:
    // Assert key down, start timing, state shared for dit or dah
    digitalWrite(ledPin, HIGH);           // turn the LED on
    ktimer += millis();                   // set ktimer to interval end time
    keyerControl &= ~(DIT_L + DAH_L);     // clear both paddle latch bits
    keyerState = KEYED;                   // next state
    break;
case KEYED:
    // Wait for timer to expire
    if (millis() > ktimer) {             // are we at end of key down ?
        digitalWrite(ledPin, LOW);       // turn the LED off
        ktimer = millis() + ditTime;     // inter-element time
        keyerState = INTER_ELEMENT;      // next state
    }
    else if (keyerControl & IAMBICB) {
        update_PaddleLatch();           // early paddle latch in Iambic B
mode
    }
    break;
case INTER_ELEMENT:
    // Insert time between dits/dahs
    update_PaddleLatch();               // latch paddle state
    if (millis() > ktimer) {           // are we at end of inter-space
?
        if (keyerControl & DIT_PROC) { // was it a dit or dah
?
            keyerControl &= ~(DIT_L + DIT_PROC); // clear two bits
            keyerState = CHK_DAH;           // dit done, check for
dah
        }
        else {
            keyerControl &= ~(DAH_L);      // clear dah latch
            keyerState = IDLE;             // go idle
        }
    }
    break;
}
// Simple Iambic mode select
// The mode is toggled between A & B every time switch is pressed
// Flash LED to indicate new mode.
if (digitalRead(cmdPin) == LOW) {
    // Give switch time to settle
    debounce = 100;
    do {
        // wait here until switch is released, we debounce to be sure
        if (digitalRead(cmdPin) == LOW) {
            debounce = 100;
        }
        delay(2);
    } while (debounce--);
}

```

```

        keyerControl ^= IAMBICB;          // Toggle Iambic B bit
        if (keyerControl & IAMBICB) {    // Flash once for A, twice for B
            flashLED(2);
        }
        else {
            flashLED(1);
        }
    }
}
//
//
//   Flash LED as a signal
//
//   count specifies the number of flashes
//
//
//
void flashLED (int count)
{
    int i;
    for (i=0; i<count; i++) {
        digitalWrite(ledPin, HIGH);      // turn the LED on
        delay (250);
        digitalWrite(ledPin, LOW);       // turn the LED off
        delay (250);
    }
}
//
//
//   Latch dit and/or dah press
//
//   Called by keyer routine
//
//
//
void update_PaddleLatch()
{
    if (digitalRead(RPin) == LOW) {
        keyerControl |= DIT_L;
    }
    if (digitalRead(LPin) == LOW) {
        keyerControl |= DAH_L;
    }
}
//
//
//   Calculate new time constants based on wpm value
//
//
//
void loadWPM (int wpm)
{
    ditTime = 1200/wpm;
}

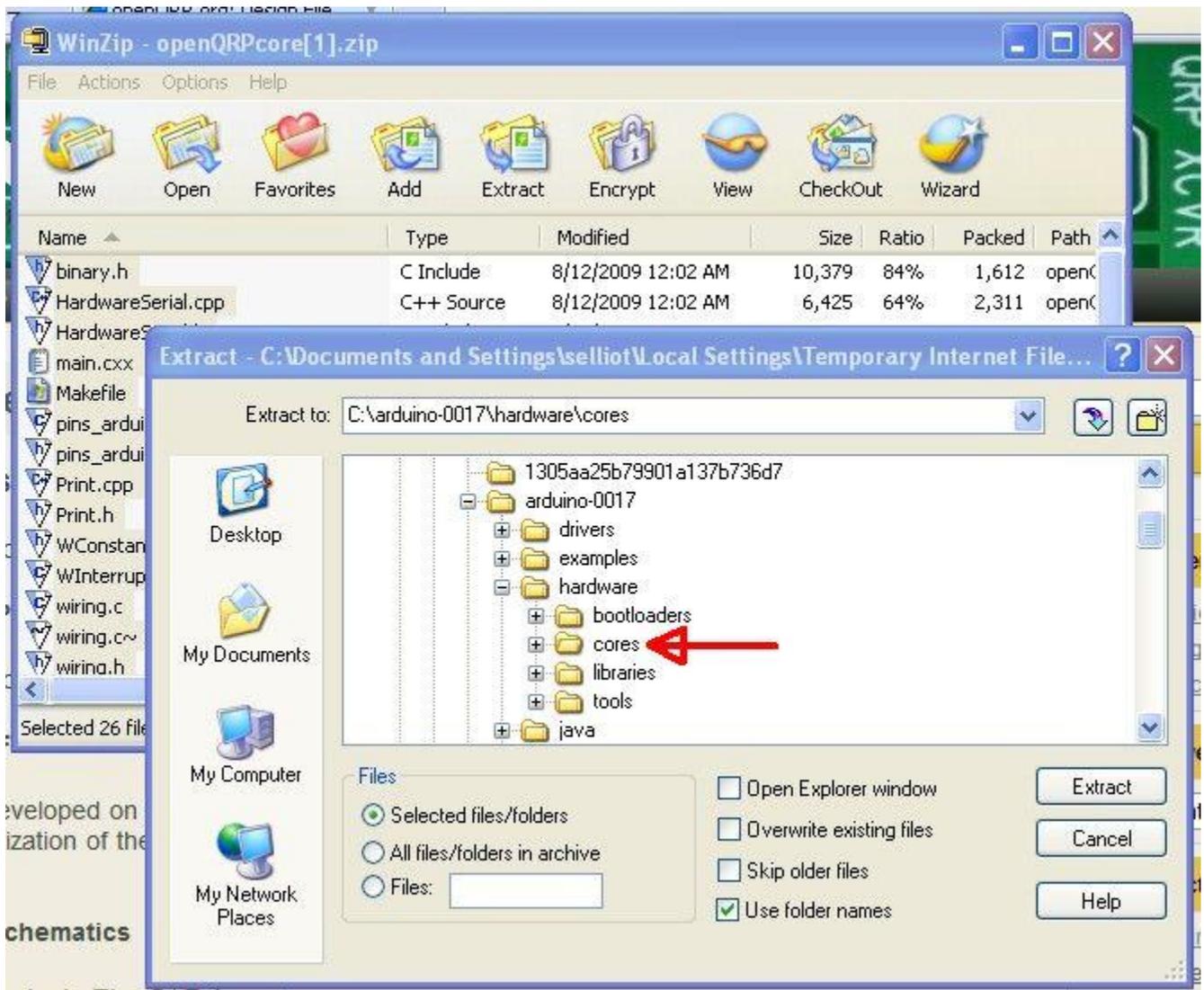
```

openQRP Arduino Files

October 21st, 2009

I posted three archive files in the Design Files/Tools area. I will discuss each one and show how to install and build a sketch under the Arduino IDE.

openQRP core: The openQRP implementation is different than a standard Arduino board in several ways. Rather than hack the existing board files I created a custom core file that contains the changes required to support the openQRP board. It's easy to install the core, simply click on the file, let WinZip open it up, select all files, and extract to the Arduino/hardware/cores directory:



You should now see the openQRP core folder as shown:



boards.txt: Now that we have installed the core we need to add an entry to the boards.txt file that describes the openQRP board. You can download the entire boards.txt file from Design Files/Tools and copy over the one in your installation. The preferred way to do this is to add the openQRP entry to the end of your existing boards.txt file because you may already have some customizations in the file that you want to preserve.

The boards.txt file is located here:



Here is the entry which you can cut and paste into your boards.txt file.

```
#####

openQRP.name=openQRP via USBtinyISP

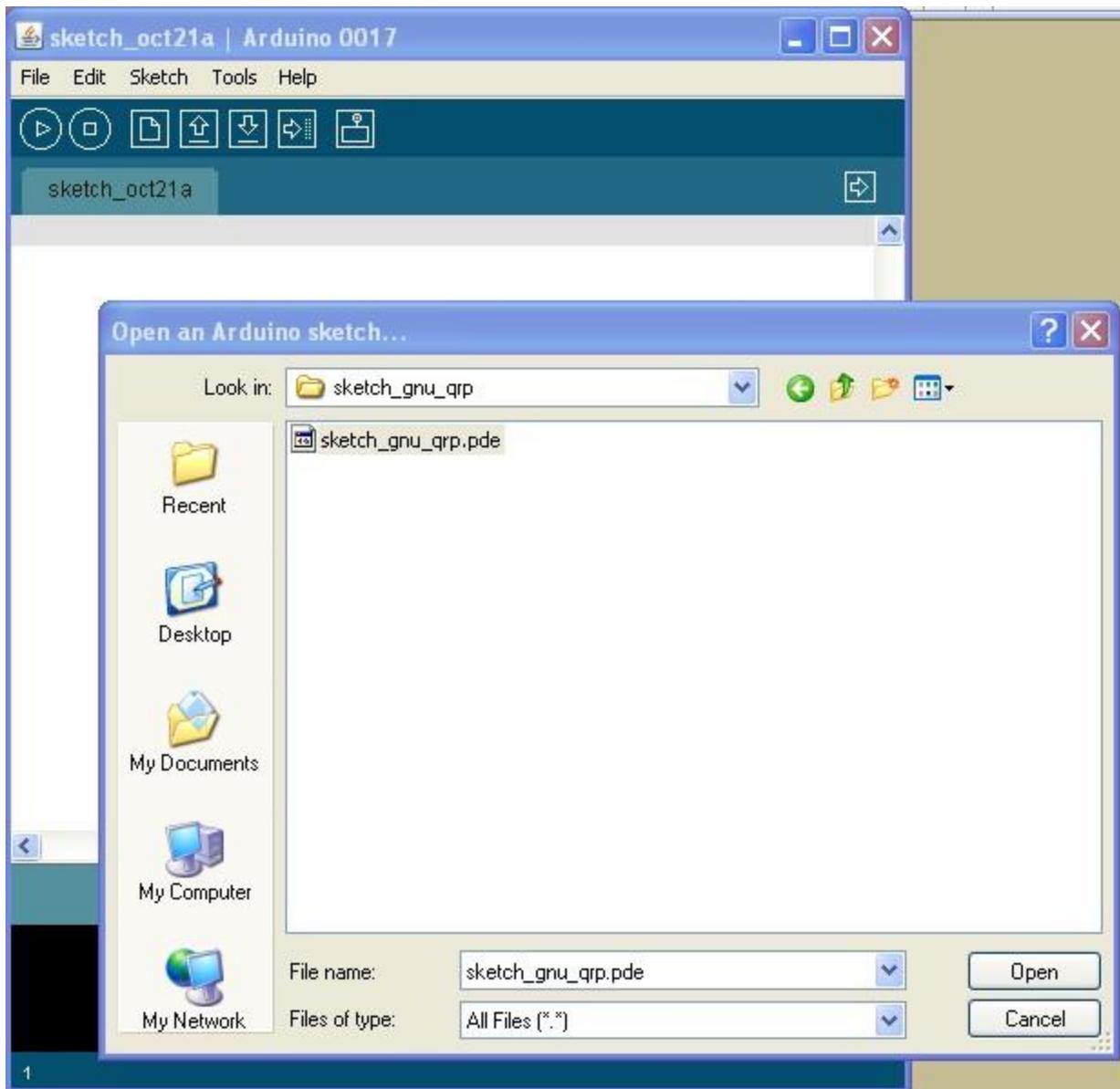
openQRP.bootloader.low_fuses=0xff
openQRP.bootloader.high_fuses=0xdd
openQRP.bootloader.extended_fuses=0x00
```

```
openQRP.bootloader.path=atmega
openQRP.bootloader.file=ATmegaBOOT_168_diecimila.hex
openQRP.bootloader.unlock_bits=0x3F
openQRP.bootloader.lock_bits=0x0F

openQRP.upload.protocol=usbtiny
openQRP.upload.maximum_size=16384
openQRP.upload.speed=19200
openQRP.upload.using=usbtinyisp

openQRP.build.mcu=atmega168
openQRP.build.f_cpu=1638400L
openQRP.build.core=openQRP
```

sketch_gnu_qrp: Finally we will install a test sketch which can be used to test our environment and make sure everything is in the correct place. First create a folder called sketch_gnu_qrp. This can be placed anywhere you like, it is not a good idea to not put it in your Arduino tools folder. I just put it in the root directory of drive C: Now open up Arduino and click File->Open and navigate to the folder you just made and select the test sketch:



Click Open. You will then see the sketch:

```
sketch_gnu_qrp | Arduino 0017
File Edit Sketch Tools Help
sketch_gnu_qrp
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stdio.h>

////////////////////////////////////
//
//   openQRP CPU Pin Definitions
//
// Digital Pins
//
int     widePin = 2;           // Wide filter select
int     STPin = 3;           // Sidetone output pin
int     LPin = 9;            // Left paddle input
int     RPin = 10;           // Right paddle input
int     narPin = 12;         // Narrow filter select
int     ritPin = 13;         // RIT & LED connected to digital pin 13
int     ledPin = 13;         //
int     antXOPin = 15;       // Antenna changeover control
int     cmdPin = 17;         // Command pushbutton input
int     keyPin = 19;         // Transmitter key output
int     lcdEPin = 18;        // LCD E control output
//
1
```

Now select Tools->Board and you will see *openQRP via USBtinyISP* select that. If you don't see an entry for openQRP, check your boards.txt file to make sure it has an entry for openQRP.

Now select Sketch->Verify/Compile which will build the sketch and if all is well you will see:

```

int pwrPin = 0; // Voltage monitor input

```

Done compiling.

Binary sketch size: 1094 bytes (of a 16384 byte maximum)

1

If you do get a bunch of errors and warnings that means the openQRP core wasn't installed in the right place.

At this point you could upload the sketch to the openQRP board and see the LED flash on and off once a second.

Let's take a look at the sketch source file. All Arduino sketches have at least three basic sections.

The first section is board specific declarations. Here names are given to all of the inputs and outputs pins on the CPU. Descriptive names are much easier to keep track of than numeric values. If you compare these names and pin numbers to the schematic you will find there is no direct correspondence. The definitions relate to [Arduino port assignments](#). There are two types of port pins Digital and Analog. Analog pins can be used as digital pins but not vice versa.

```

////////////////////////////////////
////////////////////////////////////
//
//
//          openQRP CPU Pin Definitions
//
////////////////////////////////////
////////////////////////////////////
//
//  Arduino  OQ PCB                      OQ PCB  Arduino
//  -----  -----                      -----  -----
//
//          RESET-----PC6 |1          28| PC5-----KEY_TX      Dig19 Ana5
//  Dig0    SER_REPLY-----PD0 |2  o   i 27| PC4-----LCD_E       Dig18 Ana4
//  Dig1    SER_CMD-----PD1  |3  o   i 26| PC3-----CMD_PB      Dig17 Ana3
//  Dig2    WIDE_SEL-----PD2  |4  o   i 25| PC2-----PB_NET     Dig16 Ana2
//  Dig3    SIDETONE-----PD3  |5  o   i 24| PC1-----ANT_XO     Dig15 Ana1
//  Dig4    LCD_D4-----PD4   |6  o   i 23| PC0-----MON_12V    Dig14 Ana0
//
//          VCC-----VCC  |7  o   o 22| GND-----AGND
//          GND-----GND  |8  o   o 21| AREF-----AREF
//          XTAL1-----PB6 |9  i   o 20| AVCC-----AVCC
//          XTAL2-----PB7 |10 i   o 19| PB5-----RIT_ON      Dig13
//  Dig5    FREQ_CNT-----PD5 |11 o   o 18| PB4-----NAR_SEL     Dig12
//  Dig6    LCD_D5-----PD6  |12 o   o 17| PB3-----LCD_RS      Dig11
//  Dig7    LCD_D6-----PD7  |13 o   o 16| PB2-----LPDL       Dig10
//  Dig8    LCD_D7-----PB0  |14 o   o 15| PB1-----RPDL       Dig9
//
//          -----

```

```

//                                     ATmega168
//
// Digital Pins
//
int      widePin  = 2;      // Wide filter select
int      STPin   = 3;      // Sidetone output pin
int      LPin    = 9;      // Left paddle input
int      RPin    = 10;     // Right paddle input
int      narPin  = 12;     // Narrow filter select
int      ritPin  = 13;     // RIT & LED share digital pin 13
int      ledPin  = 13;     //
int      antXOPin = 15;    // Antenna changeover control
int      cmdPin  = 17;     // Command pushbutton input
int      keyPin  = 19;     // Transmitter key output
int      lcdEPin = 18;     // LCD E control output
//
// Analog Pins
//
int      PBnetPin = 2;     // Pushbutton network input
int      pwrPin  = 0;     // Voltage monitor input
//
////////////////////////////////////
////////////////////////////////////

```

The next section is the setup function. This is where we put all of the initializations that are necessary before actually starting the main program loop. This is only called once at power up. All we need to do is specify that the LED pin is an output.

```

////////////////////////////////////
//
// System Initialization
//
////////////////////////////////////

void setup() {

    pinMode(ledPin, OUTPUT);    // sets the digital pin as output

}

```

Now we are at the main loop of the sketch:

```

////////////////////////////////////
void loop()
{
  digitalWrite(ledPin, HIGH); // sets the LED on
  delay(1000);                // waits for 1000 milliseconds = 1 second
  digitalWrite(ledPin, LOW);  // sets the LED off
  delay(1000);                // waits for 1000 milliseconds = 1 second
}

```

This function is called continuously after setup() completes. This is where all the action is. In our test sketch the loop continually turns the LED on, waits one second, turns the LED off, and waits another second. This is repeated forever. The functions digitalWrite() and delay() are built into the Arduino library, all we have to do is call them and they work. A list of Arduino functions can be found [here](#). Arduino has a great [tutorial](#) that I highly recommend.

So what exactly is the difference between the openQRP core file and a standard Arduino core ? Well not too much at this point, it's a good idea to stay as close to standard as possible. This makes life easier when we upgrade to new Arduino releases. In our core, wiring.c has been modified to adjust the timing routines for a 16.384 MHz oscillator and to add a frequency counter capability to the Timer Zero interrupt handler. Two new functions are added: getFrequency() and getTick(). Whenever functions are added to wiring.c they also must be added to wiring.h. While I was in wiring.h I also added typedefs for UCHAR, UWORD, and ULONG. We'll talk about those later.

Next time I will introduce a very simple Iambic keyer that supports both A and B modes. Then we will give firmware a rest and move on to install the LCD display and VFO sections.

USBtinyISP Programmer

October 20th, 2009

I'm going to devote an installment to the USBtinyISP programmer I use to upload sketches into the openQRP board. This is a great little programmer kit that you can buy from [adafruit](#) for \$22 plus shipping. It takes about one half hour to build and about half that time to install. There are great assembly instructions [here](#). I was unable to get the programmer to reliably program boards until I removed zener diode D1 or D2. That could have just been faulty sneers but I'll just throw it out as a suggestion.

Installation Tips

First of all you need to download the USB driver for USBtinyISP. You can get it from [here](#), make sure you get version v1.12. The download is a zip file which will automatically extract into a directory called usbtinyisp. You can tell WinZip to put the directory wherever you like, just remember where since you will need to know the path when it's time to install the driver.

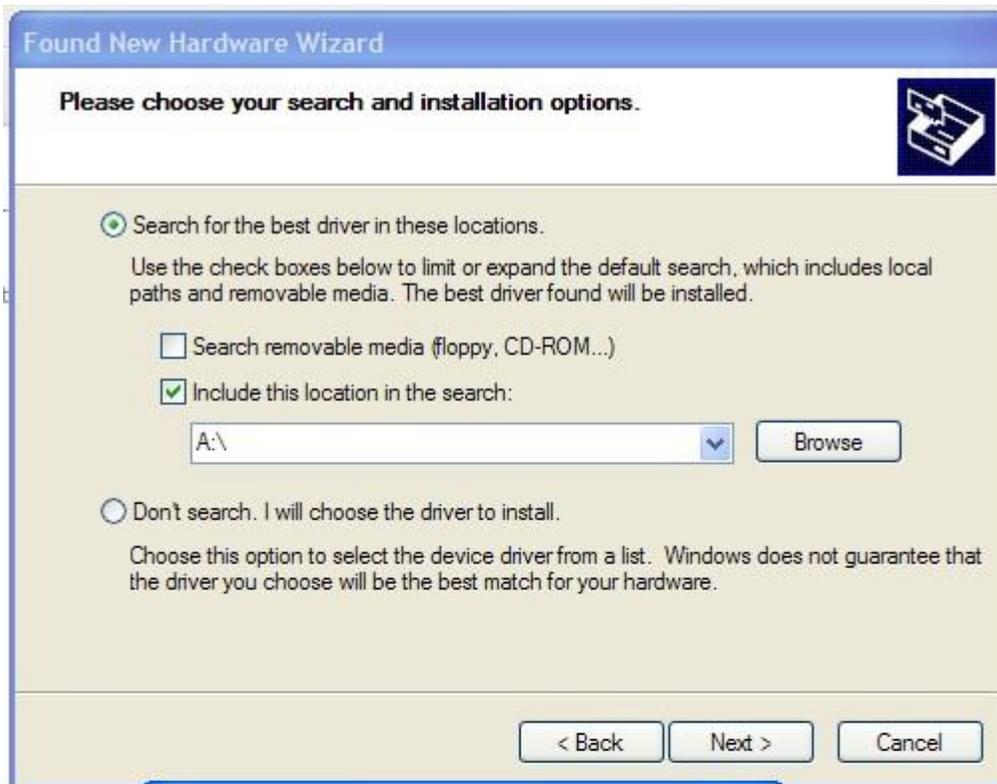
Once you have built the programmer it's time to test it out. First of all plug it into your PC's USB port and you should get this dialog box:



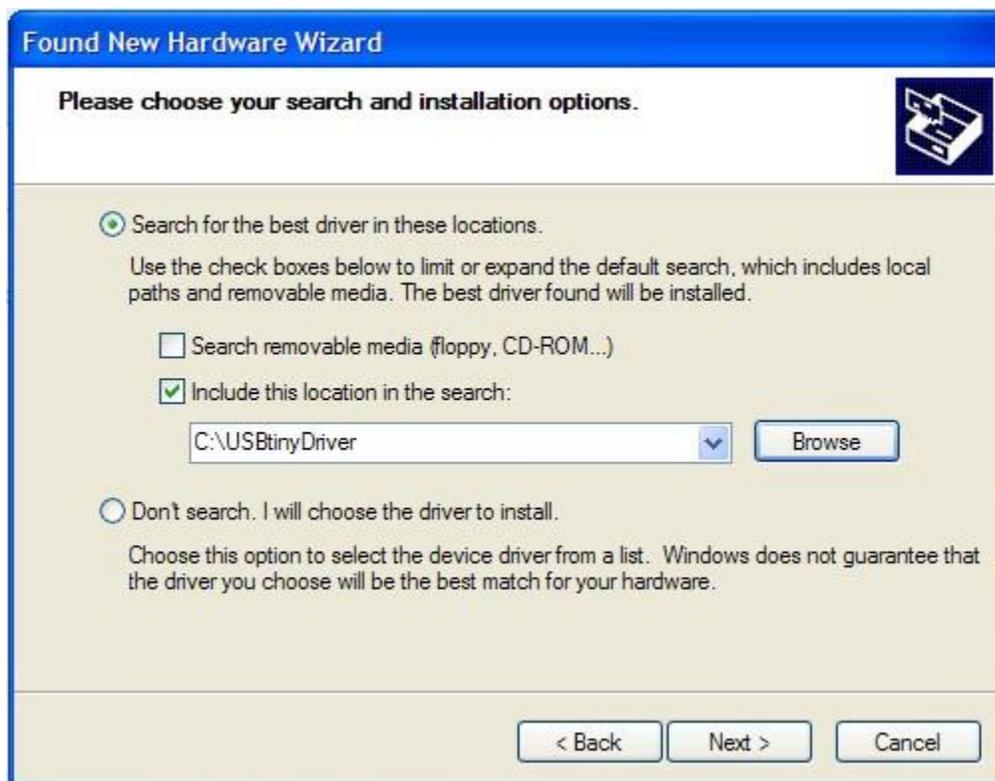
Click No, not this time and Continue. The following dialog box will then display:



Click Install from a specific location and Next:



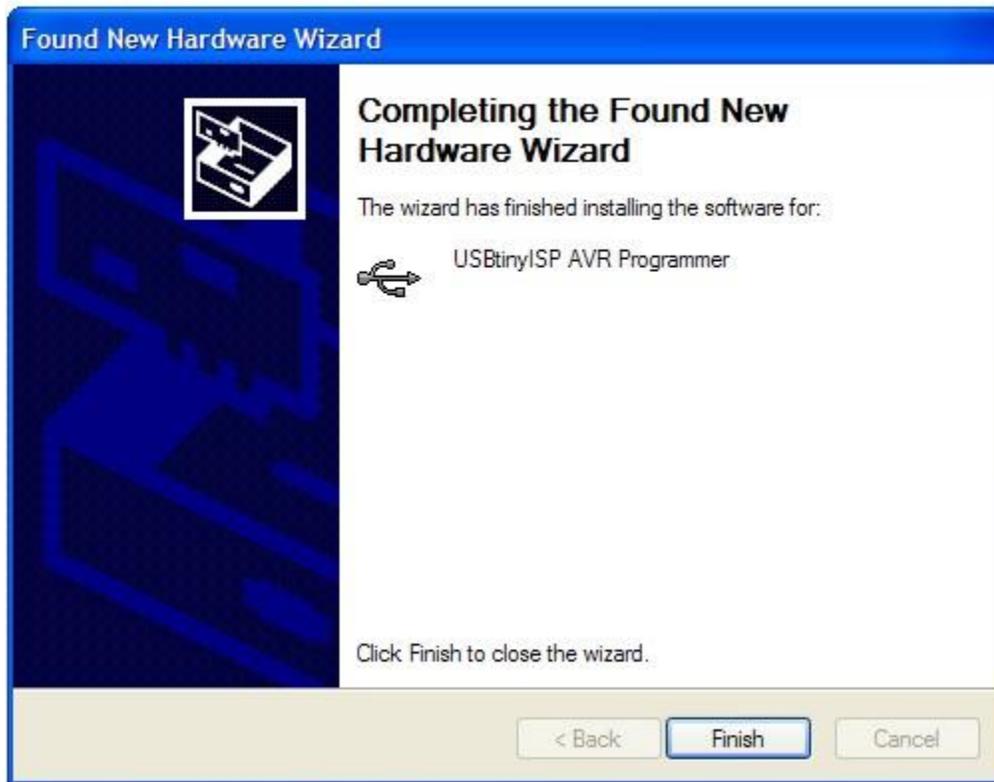
Browse to the directory that you placed the install files then click OK:



Click as shown then Next:



The driver will install and then show the following dialog box, Click Finish



Now unplug the programmer and then start up Arduino. Select Tools->Burn Bootloader->w/USBtinyISP and you should get the following message:



Now plug in the programmer and after 3 or 4 seconds Select Tools->Burn Bootloader you should then get this dialog box:

```
Error while burning bootloader.

avrdude: initialization failed, rc=-1
      Double check connections and try again, or use -F to override
      this check.
```

This means that the programmer was detected but the upload failed because the programmer was not connected to a target board. This is exactly what we want to see, the programmer is ready for the next step.

Next time we will install the openQRP customization into Arduino and then we can upload sketches.

CPU Bring Up – Continued

October 19th, 2009

There was a more of a learning curve with the CPU than I expected. I have been using Arduino microcontrollers for some time now and have quite a bit of experience with them. I started with their Duemilanove board and used that platform for basic software development. When my target board was ready I simply pulled the CPU out of the Duemilanove board, plugged it into my prototype board and continued. This was a leap from using the USB download capability of the Duemilanove board to using USBtinyISP and the ICSP connector to upload sketches to the CPU.

If you look at how most Arduino designs work, the CPU contains a preloaded bootloader that accepts an uploaded sketch from the serial port and transfers it into the ATmega168's flash memory program space. The advantage of this process is that it is very simple and easy to use. The disadvantage is that you need to include a USB to serial interface IC in your design and give up about 2K of flash code space for the bootloader. I did not want either of these things on the openQRP board, instead I opted to use an ICSP connector over which I can upload a sketch directly to flash memory without a bootloader. USBtinyISP takes care of the USB interface link and Arduino supports USBtinyISP so I'm all set. And it worked great on my initial prototypes since I always started with a preprogrammed CPU.

I now proceed with my new openQRP PCB, USBtinyISP, and an empty ATmega168 CPU fresh from Mouser. I start from scratch on a new PC, install Arduino, USBtinyISP driver, and build up the openQRP environment. I begin testing with a simple sketch that flashes the openQRP LED off and on every second. I upload that to the board and lo and behold the LED comes on and stays on..... then goes off and back on every 16 seconds. Why so slow ? Recheck my test code, all looks good. Pop out the new ATmegaCPU and plug in the CPU from the Duemilanove board, upload the test sketch into that.... guess what ? The LED flashes off and on every second ! So what is different between the two CPUs ? The Duemilanove CPU had been preloaded with

a bootloader, but since an upload through ICSP is started directly after loading, the bootloader is not used. After some reading I remembered that there are configuration fuses in the ATmega CPU that need to be programmed. A stock CPU is set up to run on an internal 8MHz clock using an internal divide by eight for a net clock rate of 1 MHz. I had expected a clock rate of 16.384 MHz, roughly 16 times greater. So that's why the LED flash rate was 16 times slower than I expected. So, how do we program the fuses ? One way is to figure out the [fuse settings](#), use [avrdude manually](#) with USBtiny, and program them directly. A much easier way is to use Arduino to download a bootloader because when you do that, fuse programming is included as part of the process. I don't care if the bootloader gets programmed, I'm not going to use it, but it is nice to have Arduino set the fuses exactly the way they should be, that gives me one less thing to worry about. And the fuses only need to be programmed once. So that's what I do. After a minute wait, the process completes, the fuses get programmed and now when I upload my test sketch the LED flashes off and on once a second !

The reason I covered all of this is that it is handy to know how to do this sort of thing if you ever want to build your own Arduino controller. All of this information can be found in bits and pieces out on the web but it takes quite a bit of work to put it all together.

In the next installments I'll cover how to install and test the USBtinyISP programmer, how to install the openQRP environment in Arduino, and how to program a target board.

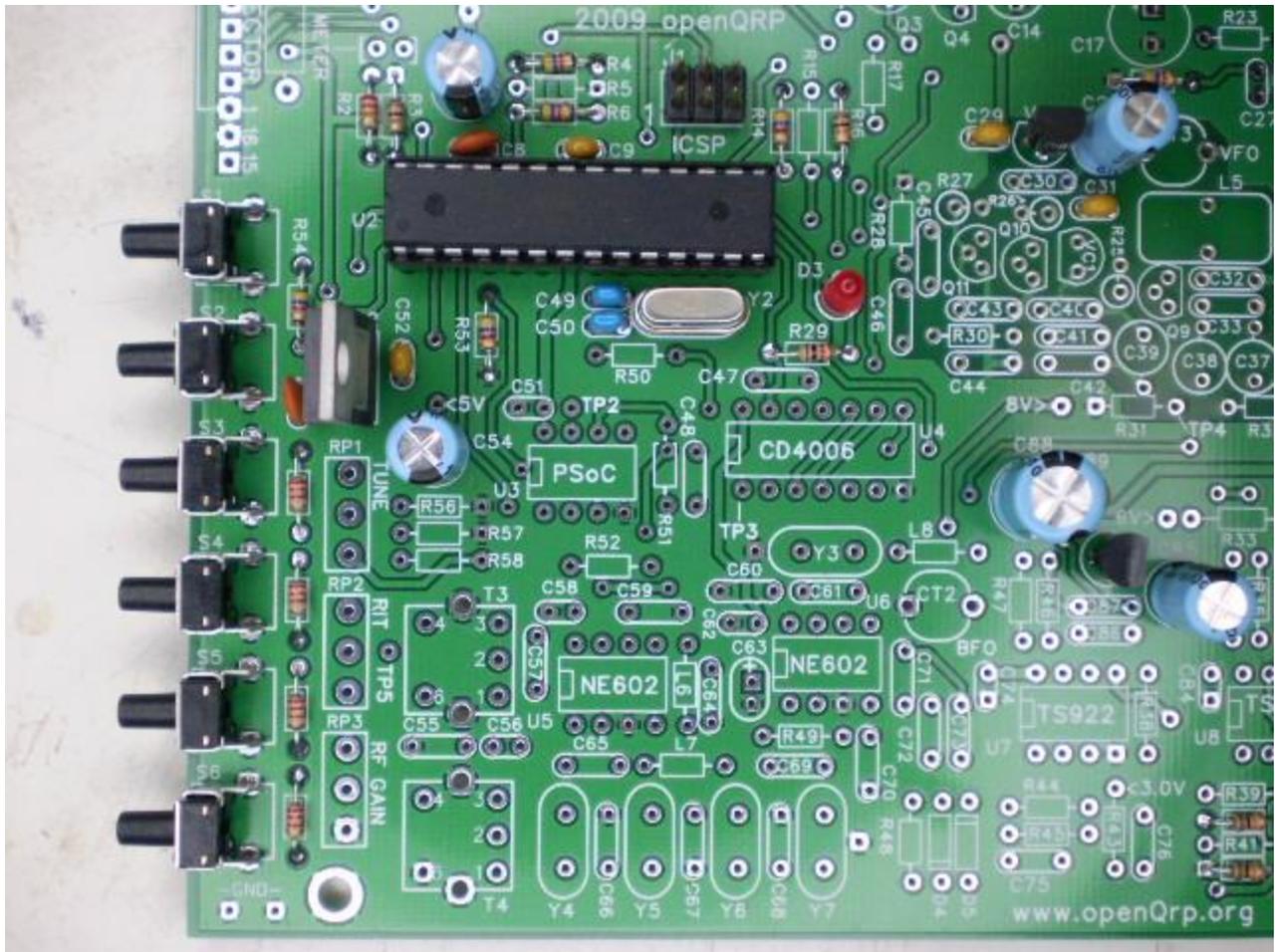
CPU Installed and Running

October 19th, 2009

I installed the Arduino CPU complex and it's operational. I have been able to upload test sketches* into the CPU through the ICSP connector and test basic functions. I'll cover the pieces of the CPU complex that were installed:

The CPU is clocked by a 16.384 MHz crystal that is directly attached to the CPU. An internal clock oscillator resides inside the CPU. The CPU has many inputs from different sources on the board. The command pushbutton and paddle inputs have pull-ups to assert these inputs high when the switches are not closed to ground. Five additional pushbuttons are connected to resistors making up a multi-tap voltage divider, when one of these pushbuttons are closed the voltage applied to a CPU input changes which is read by an analog to digital converter inside the CPU. This arrangement allows us to read five pushbuttons with one CPU input pin. By assigning the command pushbutton its own separate input, we can detect when two pushbuttons are pressed (command plus one of the five) to double the number of pushbutton states.

The ICSP connector has six pins, two go to power and ground, one goes to CPU reset and the remaining three are used to serially upload sketches to the CPU's flash memory. These three programming pins are shared with CPU control functions, namely RIT ON, Narrow Select, and LCD Register Select. These are all high impedance lines and are virtually invisible to the programmer during ICSP programming. The RIT ON line also drives an on board LED, this mimics Arduino designs, providing something simple to turn on and off for our initial CPU tests. Not shown in the picture is the paddle input connector, I installed that so I could plug in a set of paddles and write a simple CW keyer that will light the LED.



I had a bit of trouble getting the CPU operating at the correct frequency, this is the first unused ATmega168 CPU I have used and I learned that there are configuration fuses that have to be programmed first. I'll cover this in the next installment tomorrow.

**sketch* is an Arduino term for a project file that is edited and compiled in the Arduino IDE and then uploaded to the board. It is a much more friendly term than object files, code, source files, etc.

Power Checkout

October 13th, 2009

A beautiful weekend here in NH, hiking and family took priority over the OQ board. I did manage to install and check out the system power supplies. After doing a short and open test I always like to verify all power supply sections first. It's much easier to debug power problems at the beginning. Luckily all rails checked out fine. There are four main system voltages:

Five volt digital power: CPU, PSoC, VFO sampler, LCD Display

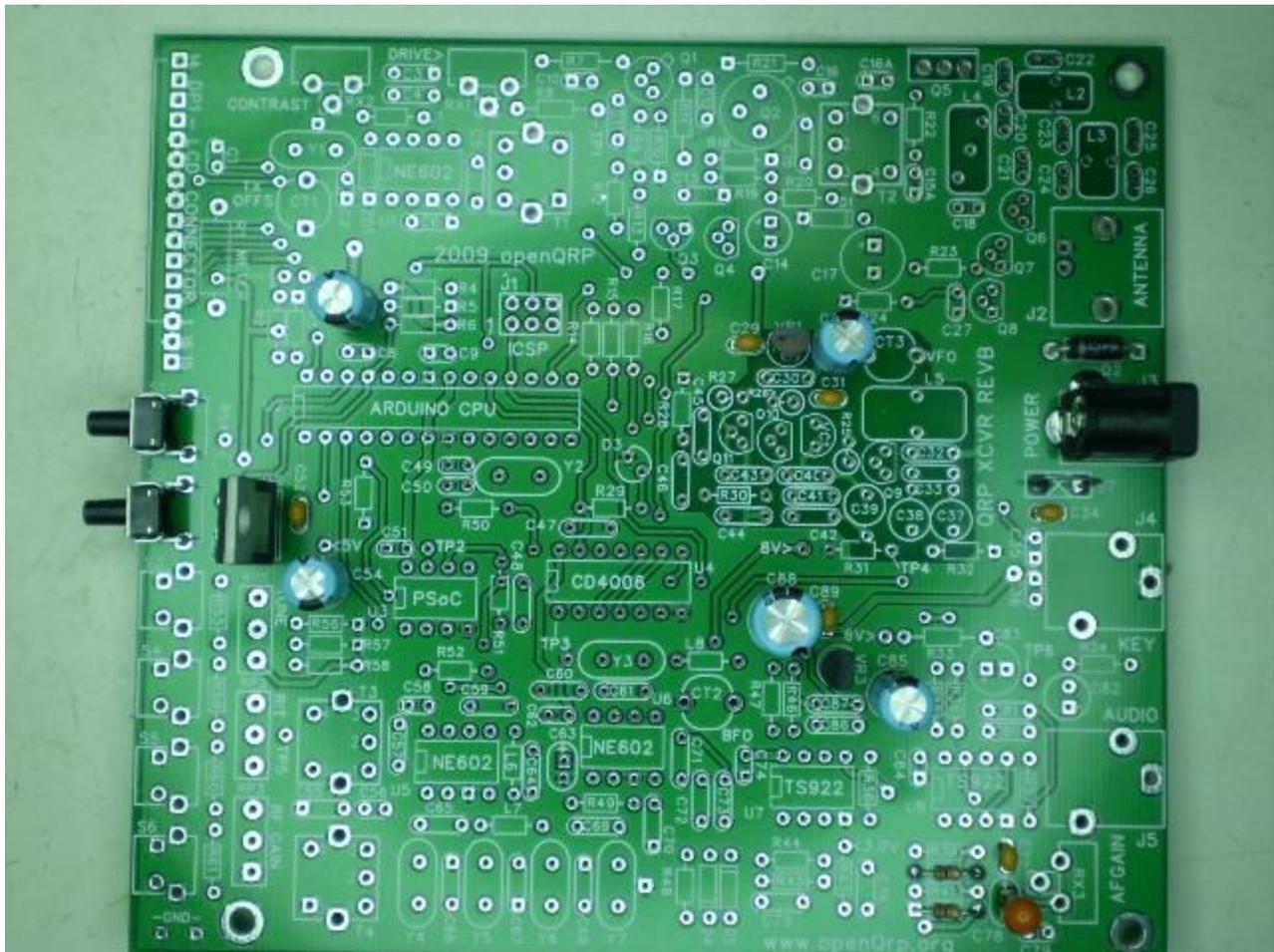
Eight volt analog: VFO, Tune, RIT

Six volt analog: 1st and 2nd Rx Mixer, Diff amp, Active Filter, Audio amp, Analog Switch

Three volt analog reference: Diff Amp, Active Filter, Audio Amp

The transmit mixer and driver stages run off Keyed 12 Volts while the Tx final runs directly off the 12 volt rail. We will test keyed 12V later.

I really like three terminal regulators, a bit more expensive than zener diodes but worth the extra quarter. In addition to being fairly load insensitive, they have built-in over current protection. (That said, I did use a zener diode regulator for the transmit mixer since it is the only thing on that circuit) I used three regulators in all, a 7805 (TO-220 package) for 5V, a 78L08 (TO-92) for 8 volts, and a 78L06 (TO-92) for 6 volts. As long as you follow the rules for capacitor placement these regulators provide very clean and stable voltage rails.



An on board power connector is well worth the real estate it takes up, off board connectors float around and eventually short against something while you are debugging and are a big pain in general. (I guess that can be said for all connectors now that I think of it) So once I had everything installed I simply plugged in my power cable and quickly checked all the test points. The picture shows two pushbuttons on the board, not soldered in, just there to make sure the PCB footprint is correct. The most stressful thing about a new PCB is the whether all the footprints are correct and oriented properly. On the previous OQ board spin I placed the

pushbutton switches backwards ! No clean way to recover from that sort of error, instant scrap. Good news is that all footprints in this version are right so it's clear sailing.

You might wonder what D2 (1N4001) is for, it's a polarity protection diode. If you accidentally connect power the wrong way (plus and minus reversed) D2 is forward biased, draws a lot of current and pops the fuse in your power lead instead of electronics on the OQ PCB. It requires that your power lead has an inline fuse holder (which is always a good idea anyway). Some designs put the diode in series with the power lead so that it will only conduct current when the power leads are hooked up right, but this incurs a .7V drop in supply voltage (.7 watts at a current draw of 1A), a consideration if you're running on battery power. Either scheme is equally good just as long as there **IS** some sort of protection, it is very easy to hook power up wrong if you are in a hurry or it's late at night during Field Day....don't forget extra fuses 😊

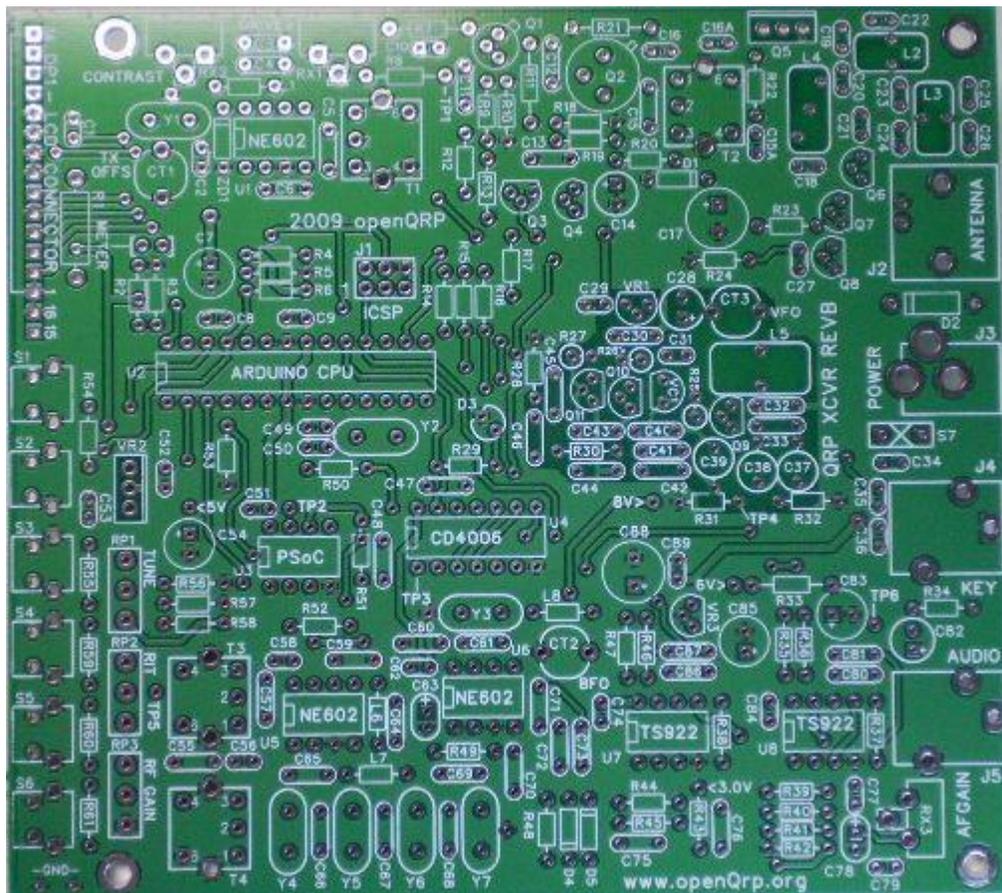
I am working on a bill of materials that I will post soon. I am breaking the parts up into groups per debug session. It will be a big help as we move along.

Next on the trail is the CPU, I'll start that tonight. That is where the fun really starts.

openQRP PCB 1st Look

October 10th, 2009

The board arrived today, here's what it looks like. I will start working on it this weekend.



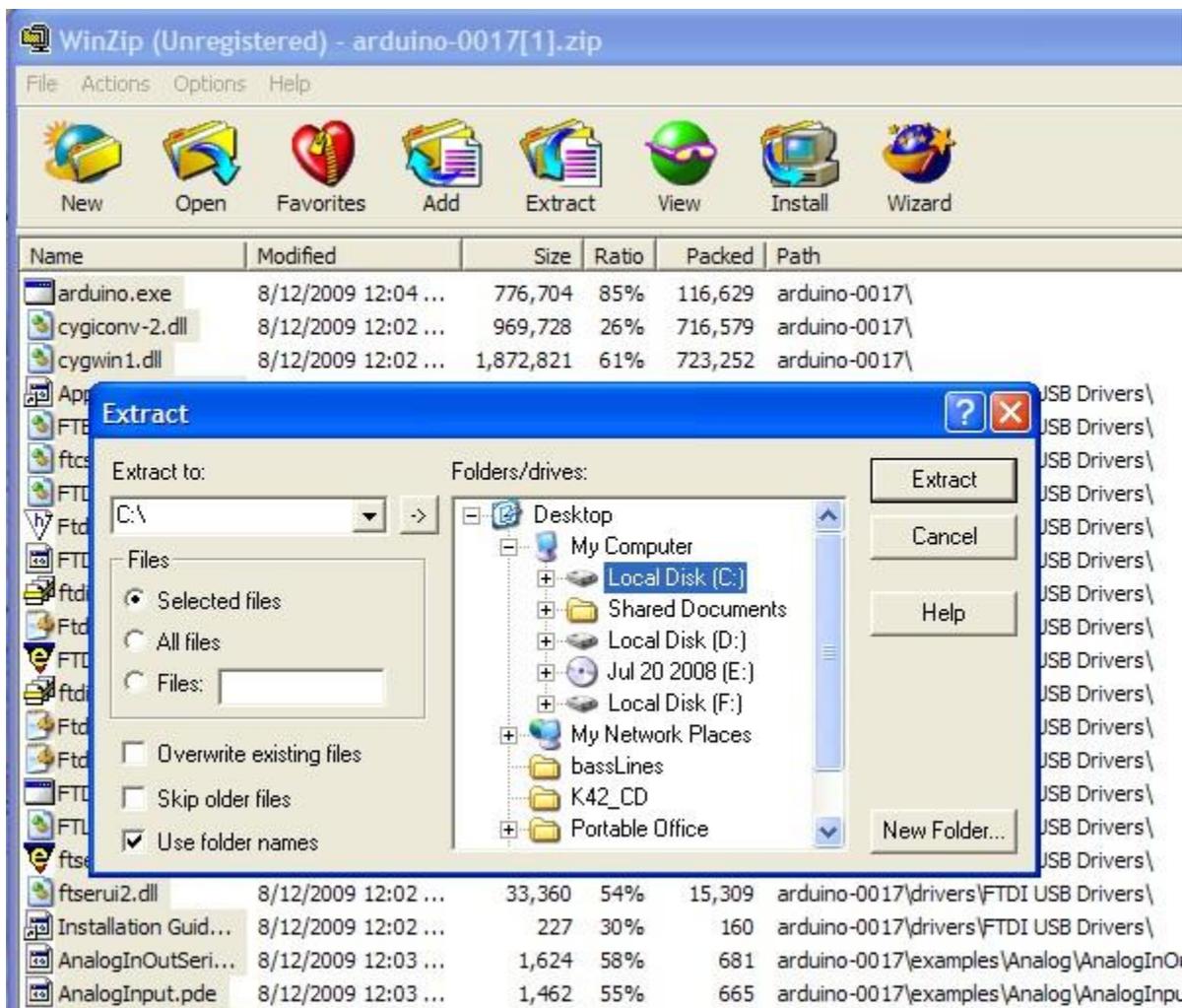
More Arduino

October 9th, 2009

Installing Arduino is very easy to do but it is not a completely automatic process. As mentioned before, we are using Arduino-0017 which you download from

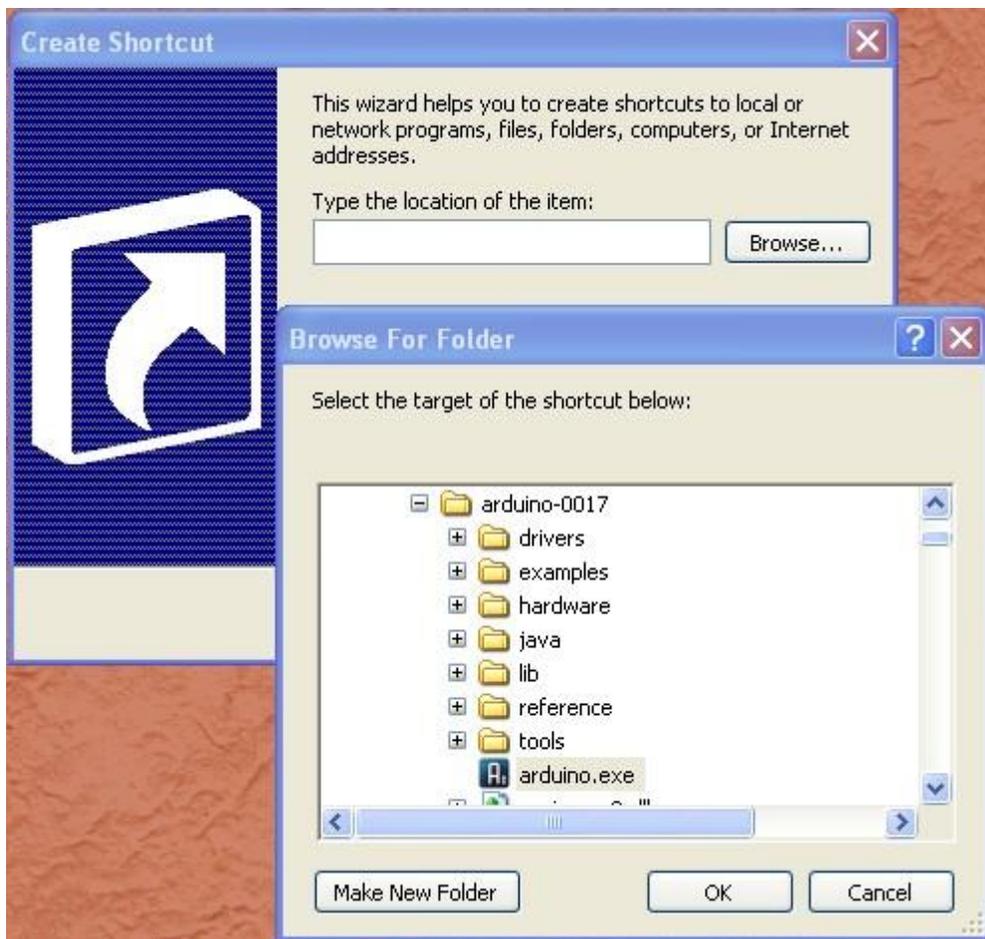
here: <http://arduino.cc/en/Main/Software>

I am using the Windows version, other operating systems are supported but I'm sticking with Windows for now. The file you download is a zip file, 80+MB in size. When the file has been downloaded WinZip will automatically open displaying all of the files in the archive. Next Click Edit->Select All Then select the hard drive you want to install to, I pick drive C:, then click on Extract and the toolset will be installed into a directory called c:\arduino-0017 (Make sure that "Use Folder Names" is selected)



To make the application easy to find, I add a shortcut and place it on my Desktop:

Right click on your desktop, select New->Shortcut and Browse to C:\arduino-0017\ and select arduino.exe



Click OK and now you will find the audio icon on your desktop which you can click to bring up the IDE.



Take a look at the file structure of the Arduino installation:



The hardware directory contains a file called boards.txt. We will be adding an entry to this file that describes the openQRP board:

```
#####  
  
openQRP.name=openQRP via USBtinyISP  
  
openQRP.upload.protocol=usbtiny  
openQRP.upload.maximum_size=16384  
openQRP.upload.speed=19200  
openQRP.upload.using=usbtinyisp  
  
openQRP.bootloader.low_fuses=0xff  
openQRP.bootloader.high_fuses=0xdd  
openQRP.bootloader.extended_fuses=0x00  
openQRP.bootloader.path=atmega  
openQRP.bootloader.file=ATmegaBOOT_168_diecimila.hex  
openQRP.bootloader.unlock_bits=0x3F  
openQRP.bootloader.lock_bits=0x0F  
  
openQRP.build.mcu=atmega168  
openQRP.build.f_cpu=1638400L  
openQRP.build.core=openQRP  
  
#####
```

Several things to notice, we specify the programmer we will be using, namely USBtinyISP. Next we specify the CPU as an ATmega168 with 16384 bytes of code space available. Since we will not be using a bootloader, the entire code space can be used for our application. Even though we aren't specifically using a bootloader we still maintain the bootloader entries. Finally we specify that our board will use the openQRP core. This is a set of definitions and low level functions that have been customized for openQRP. This will be contained in a directory called openQRP in the hardware/cores directory, we will install this ourselves.

Happy Thanksgiving !!

November 26th, 2009



Transceiver is complete, frequency counter works as well as signal strength indicator and CW reader !

I have lots of writing catch up to do. It's a great rig, listen for me on 7060 KHz....

Frequency Counter

November 20th, 2009

No pictures today. The last couple of days I spent hooking up controls and testing out more parts of the design.

The biggest step was adding the frequency counter display to the LCD.

The ATmega68 CPU contains a built in 16 bit counter that is clocked directly by a buffered version of the VFO output. This counter essentially free runs and is "gated" by an accurate millisecond interrupt routine. Since a 16 bit counter can only count up to 65535 and we need to count over 2 million cycles, we use a little trick.

As mentioned, we are counting the VFO which has a frequency somewhere between 2.084 and 2.164 Mhz. If we want to count to a 1 Hz resolution, we will gate the counter for 1 second. At the highest frequency, our 16 bit counter will overflow $2164000/65536 = 33$ times. To get around this we gate the counter for a second but check it every millisecond to see if it has overflowed. We keep a tally of the overflows and at the end of the one second gate, we multiply the overflows by 65536 (10000 hex) and add that to the count in the 16 bit counter. That gives us our accurate frequency counter. In the final user interface we will tweak this setup to display at 100 Hz resolution, that is sufficient for tuning and will give us a faster update every 10 milliseconds instead of every second.

The interrupt routine is presented below. This code resides in wiring.c which is in the cores/openQRP directory. The current frequency count in cycles per second (Hz) is kept in the variable f_freq. Every millisecond, the timer zero overflow bit is checked and if it is set the overflow tally, f_ovf, is incremented. At the end of one second the 16 bit counter is read and then added to (overflow * 0x10000) to arrive at the frequency count, the variables are then zeroed out for the next capture. The one second interval is timed using the variable f_period which is incremented every millisecond. When it reaches 1000 (1000 milliseconds = one second) the frequency count is tabulated. The variable bTick and timer0_millis are not used by the frequency counter, these are used for other timing purposes.

Timer Zero Interrupt Routine:

```
unsigned long f_freq;
unsigned long f_ovf=0;
unsigned int f_period=0;
unsigned long timer0_millis = 0;
char bTick=0;
/////////////////////////////////////////////////////////////////
//
// Timer0 Compare interrupt vector handler
//
// called (16,384,000/64/256) times per second(1000 Hz or 1 millisecond)
//
/////////////////////////////////////////////////////////////////
//
SIGNAL(TIMER0_OVF_vect)
{
    // Snap Frequency counter value every second
    if (f_period > 999) {
        f_freq = TCNT1;
        TCNT1 = 0;
        if (TIFR1 & 1) {
            f_ovf++;
            sbi(TIFR1, TOV1);
        }
        f_freq += (f_ovf * 0x10000);
        f_period = 0;
        f_ovf = 0;
    }
    else if (TIFR1 & 1) {
        f_ovf++;
        sbi(TIFR1, TOV1);
    }
}
```

```
f_period++;
timer0_millis++;

// bTick is the one millisecond semaphore that kicks off loop() tasks
bTick=1;
}
```

Once the frequency counter and display were working I could then calibrate the VFO and adjust its range to start exactly at the bottom of the 40 meter band, 7.00 MHz. I also tweaked the 2nd Rx mixer to align the crystal filter peak response to be at 690 Hz. I did this by padding C61 to 100 pF. This gives a much more useful adjustment to CT2.

Before starting on VFO calibration, the tune and RIT controls were attached. Now that I have a front panel this is very easy. In previous versions of this project the controls would just hang off the front of the PC board, a recipe for shorting and wire breaks.

I added some pushbutton handlers to the firmware to allow Wide/Narrow and RIT ON/RIT OFF to be selected. My RIT range is not quite centered as I designed, this will take a tweak of a value in the RIT bridge circuit. I'll cover the way RIT works in the next installment. I still have to go back and cover the crystal filter design too... lot's to do here !

So here we are at the weekend, the next thing on the list is to align the Tx offset in the 1st Tx mixer so that we transmit on the same frequency we are listening on. I am going to work on a calibration routine in firmware to help with this. Then I will finally be able to make some contacts with this radio !

I still have to do a little bit of gain redistribution in the receiver chain. I need to adjust things so that the sidetone level is compatible with average signal level. Very strong signals don't clip in the diff amp until they are earsplitting, so I need to increase the diff amp gain and lower the AF gain level. It's a bunch of experimenting. I had it balanced nicely in the previous PCB version so I have to look at what I changed to get it right again.

One final task is to install the PSoC IC and get the CW reader and signal strength meter working. Once that is done the basic transceiver checkout will be complete.

Chassis Base Arrived

November 17th, 2009

I received a prototype of the enclosure base on Friday. It has a couple of bugs, this always happens with 1st metal. With a little filing and drilling I was able to fit the board to the metal. Now that I had the board mounted, I could then attach the LCD display and verify that it works. I got a couple of display connections wrong on the previous PC board so I was anxious to check out this new version to make sure I fixed it correctly. Now I can test out the CPU based frequency counter and also hook up the tuning and RF gain controls. I'll leave the RIT for later.





Receiver Section Installed and Operational

November 13th, 2009

I find that receivers are the most interesting part of a transceiver. There's something remarkable about being able to actually pull a station out of the air, tune it in, and listen.

This is a basic NE602A design, been done a hundred times, but it works well and is easy to reproduce. It's a superheterodyne which means that an incoming RF frequency is mixed with a variable frequency oscillator to convert it to a common intermediate frequency (IF). The signal is then passed through a filter that has been optimized at the IF frequency. In this design the signal is mixed a second time to convert the IF frequency to audio that we can listen to. We call this a double conversion superheterodyne (or superhet for short). Since the chosen IF frequency is 4.9152 MHz, we mix the incoming RF with a VFO frequency of 2.084 MHz – 2.164MHz to provide a tuning range of 7.0 MHz to 7.080 MHz which is most of the CW portion of the 40 meter amateur radio band.

Let's trace a signal from antenna to headphones. Signals from the antenna are fed in reverse through the output PI filter which acts a low pass filter. It is then coupled through the Rx/Tx changeover FETs to the input bandpass filters T4 and T3. These are 10.7 MHz IF transformers that are padded down to 7 MHz with 56 pf capacitors. Two transformers are used, slightly

stagger tuned to give a wider response across the CW band, much better than a single filter could. The input signal is fed into the low impedance winding of the first transformer which mates up nicely with the 50 ohm antenna input. These IF transformers have many advantages, they are small and rugged with built in shielding and since they are slug tuned they can easily be adjusted to resonance. The high impedance secondary is a parallel tuned circuit which is lightly coupled into the high impedance tuned coil of the second IF transformer T3.

The first NE602A mixer has an input impedance of about $1.5K \parallel 3pF$ which is about 1200 ohms at 7 MHz. Rather than take output from T3's secondary, which is very low impedance, signal is taken at the 5th tap of the tuned coil to give us a better match. Using the tap as a pickoff point instead of the top of the coil helps to preserve the Q of the tuned circuit. Granted, the signal level is not as high but this will pay off in terms of better selectivity.

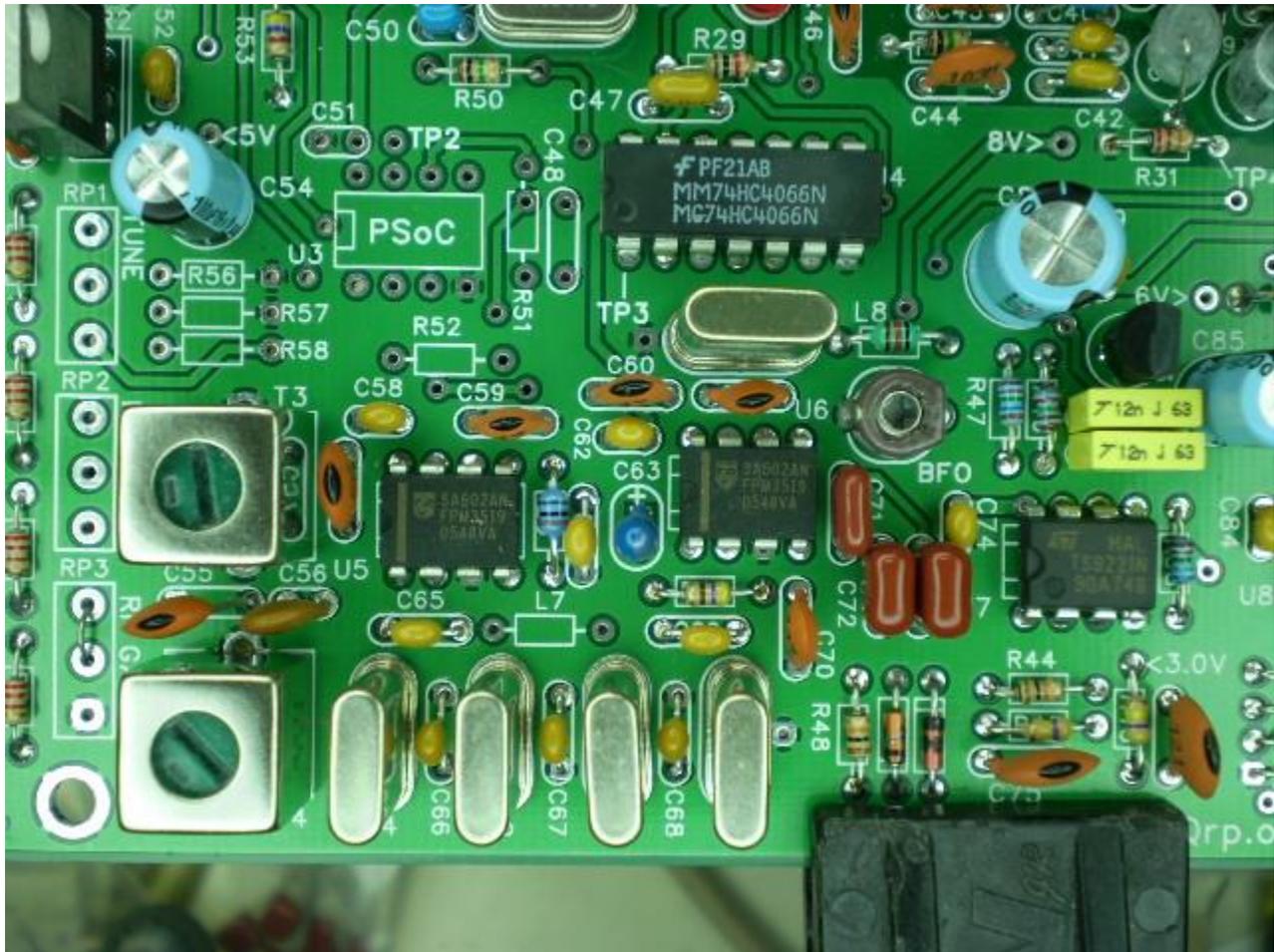
The first NE602A, U5, mixes the incoming RF with VFO energy to produce a sum product at 4.9152 MHz which is passed through a 4 pole crystal ladder filter. The design of this filter will be covered in the next installment. The basic specification is 340 ohm input, 550 Hz bandwidth, and 340 ohm output impedance. Only one leg of U5's dual output port is used and it has an output impedance of 1500 ohms. An LC matching network is used to match this to the 340 ohm crystal filter input. The calculated matching network values turned out to be 20 uH in series with 39 pF shunt to ground. Since 20 uH inductors seem to be very difficult to find (for me anyway) I chose to use two 10 uH in series. If I can find 20 uH someday I'll just install a jumper for one of them. It's nice to have some flexibility particularly when through hole components are involved.

Output from the crystal filter is fed directly into one input port of the second NE602 mixer U6. A 470 ohm resistor in parallel with the 1200 ohm mixer input gives 337 ohms which is a very good match to the crystal filter's design output impedance of 340 ohms. Since the input frequency to the second mixer is always 4.9152 MHz, a fixed crystal local oscillator can be used as the mixing source. The crystal frequency is the same as the IF frequency, 4.9152 MHz, with padding inductance and capacitance to allow the oscillator to be pulled off frequency by 690 Hz. Thus the resulting mixing product of the oscillator and a signal in the center of the filter's passband will be 690 Hz. This is an important frequency since the two stage active filter and the PSoC CW reader (both downstream) are set to this.

The second mixer provides a differential output which is fed into a high gain differential amplifier U7A. Differential amplification has many advantages particularly for high gain stages. Only the difference between the two outputs of the NE602A are amplified, that means power supply and other common mode noise is rejected. The gain for this stage is set at about 47. High frequency filtering is accomplished just before the diff amplifier by a .01uF capacitor across the output of the second mixer. In addition, two 120pF caps in the negative feedback path of the diff amp lowers the high frequency cutoff, making this stage a low pass filter. This is essential to attenuate the high frequency products produced by the 2nd mixer, this is heard as hiss in the output.

Following the diff amp is a two stage active filter providing 4 poles of selectivity, the overall design bandwidth is 300 Hz for both stages and about 400Hz for one stage. The CD4066 analog switch U4 is wired so that one or two stages can be selected by the CPU output pins. No matter

what bandwidth the user selects, both stages are always used to filter input to the PSoC. Output from the filter is coupled to the final AF amp through the AF gain trimmer. The final amplifier has been talked about before, it has a fairly low gain, 10, and feeds a set of stereo headphones in a special way. The load is applied across the left and right side of the headphones so that the speaker elements are in series. The ground is not connected. This results in a higher output impedance which reduces the amplifier drive requirements. If someone does plug in a set of mono headphones it still works ok since the mono plug will short the ring connection to ground and all is well.



Completed Receiver Circuit

This circuit is fairly easy to align and test. An antenna is connected to the BNC antenna jack and a set of headphones and power are attached. With power on I could hear a faint hiss in the headphones. I adjusted T3 and T4 to peak this noise and then adjusted the local oscillator crystal padding cap CT2 to peak the noise frequency to match the filter passband. By then tuning around I heard several stations coming in loud and clear. I then fine-tuned the adjustment to get best results.

To quickly verify the crystal filter performance I attached a signal generator set to 7.020MHz to the antenna connector. After tuning the receiver to the generator frequency I then swept the

generator frequency across the filter and plotted the response. I used the output of the diff amp as my data collection point and found the filter -3dB passband to be 520Hz. I also experimented with the crystal filter input matching circuit by shorting one of the 10 uH inductors. With both inductors installed for a 20 uH value, the filter response was very flat in band with nice steep slopes at the edge of the passband. With 10 uH the passband response was not nearly as flat and the slopes edges were not as symmetrical.

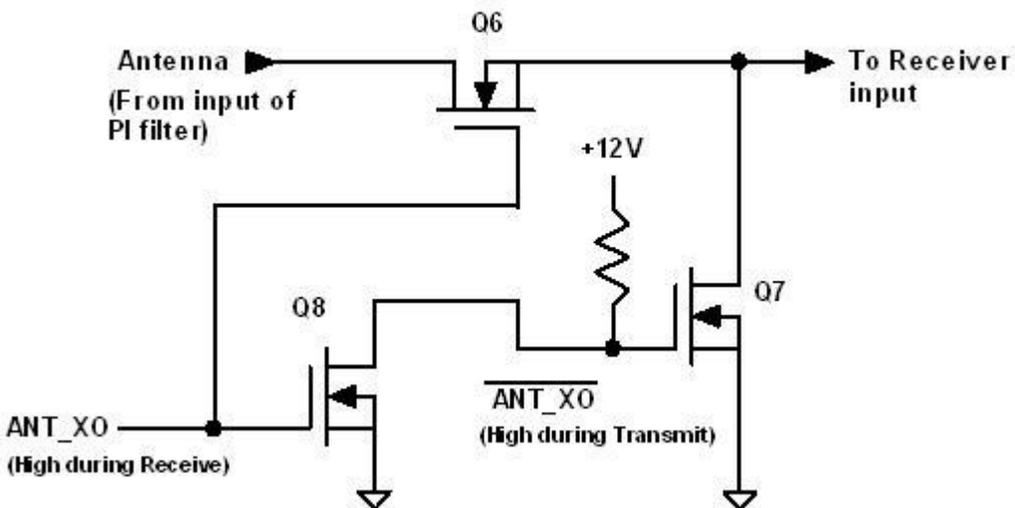
The audio level of the receiver is not quite what I had hoped for, I think my local oscillator may require more adjustment to pull from passband center instead of slightly off which I think I am now. In addition, I may need to add slightly more gain in the diff amplifier. I would rather put gain here than in the audio amp stage so as not to degrade the signal to noise ratio.

But all in all it works very well.

Antenna RX/TX Change Over Switch: Design Revision

November 9th, 2009

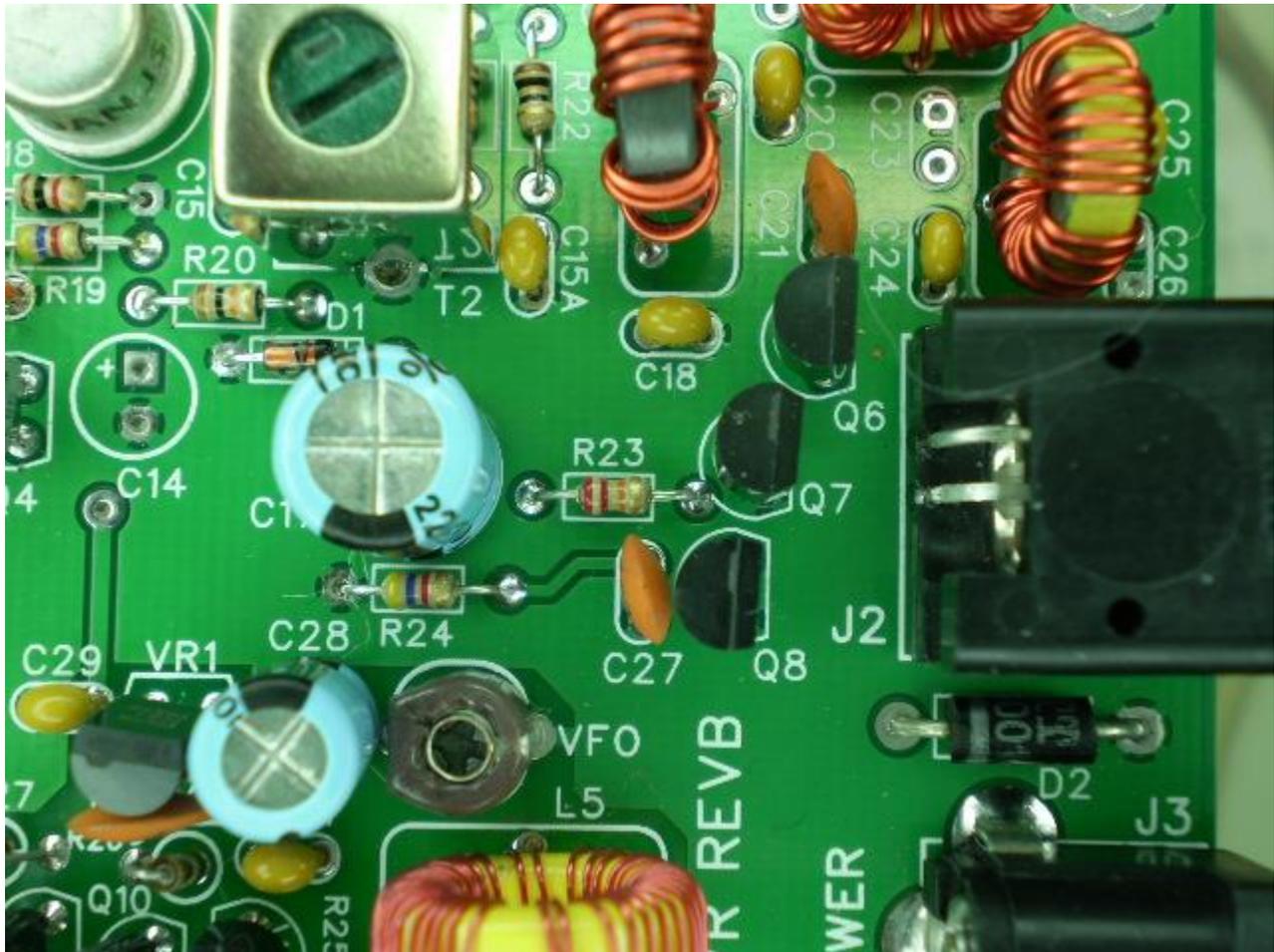
In any transceiver, the receiver and transmitter share the same antenna connection. We need a way of disconnecting the receiver from the antenna when we transmit. To prevent damage, we don't want to apply the high voltage output of the transmitter directly to the sensitive receiver front end. Three MOSFETs are utilized to do this. I have seen this used in many designs, particularly by Steve Weber. It's a clever way of Tx/Rx switching without the use of a relay or switching diodes. In writing this installment I realized I had a design flaw. The new transmitter design produces about 8 watts out which translates to about 56 volts peak across the drain to source junction of the IRF510 final amp. This is at the input to the output PI network where we pick off the RF feed to the receiver. The IRF510 is able to handle up to 100V Vds so that is ok. But if you look at the antenna changeover switch:



In transmit mode, Q7 is turned on while Q6 is open. This is a good thing since it isolates the receiver input from the transmitter output. It also ties the receiver input to ground as an extra safety measure. Bad thing about this: the full final voltage of 56 volts is applied across the Drain to Source junction of Q6. This is fine for a perfect match into 50 ohms but as soon as we

stray from a purely resistive to a reactive load, in other words SWR not equal to 1:1, the voltage can increase to easily reach 70-80V. While the IRF510 can tolerate this, Q6 will fail as soon as we go over its maximum allowable V_{ds} of 60V. Luckily, there is a good replacement for Q6, a ZVNL110A, which has a V_{ds} max of 100V and is available in the same TO92 pinout as the 2N7000. It's about twice as expensive, (50 cents vs. 25) but that is a small price to pay for reliability.

You can't tell what will happen when a transistor junction fails, it can either open or short. Open we can live with but a short will surely damage the receiver. In any case we want to design to prevent failure.



Rx/Tx Changeover components populated

Next installment will cover the receiver circuitry followed by the design of the crystal ladder filter.

Active Audio Filter and AF Amplifier

November 6th, 2009

I'm still waiting for the metal chassis plate, so I'm working piecemeal on other portions of the radio. I installed the AF headphone amplifier and the four pole active filter. The AF amplifier has gain trim control, I find that it's more useful to have an RF gain control and trim the AF gain to a comfortable level for sidetone and leave it there. The RF gain control is used to compensate for varying signal level. This design does not use AGC, it's not necessary for reasons I'll cover in a later installment.

The active filter is two stage, two poles each. Dual op-amps are used for these, the first stage is shared with the recovered audio differential amplifier, and the second stage is shared with the headphone amplifier. This is a good grouping as signal levels are compatible, less chance for crosstalk. The second op-amp power supply has additional filtering through a 10 ohm resistor and local 100 uF filter cap. This limits very large audio swings to protect your ears and also reduces noise on the 6 volt bus. I also installed the quad analog switch since this is part of the audio chain. It has two primary functions, select between one and two stages of active filtering and disconnect audio for muting purposes during transmit. Sidetone is generated by the CPU, at the center frequency of the active filter, and injected into the audio chain through R50, a one megohm resistor, during transmit. The CPU generates a square wave sidetone signal which, after passing through the active filter is made more sinusoidal and sounds pretty good. The sidetone signal provides a good way to debug the active filter and AF amp functionality.

The firmware has grown considerably, now there is capability to generate Morse letters from ASCII characters, as well as transmit keying and muting. I'll post a new firmware version tonight.

This weekend I plan to finish up the receiver and actually try to make some contacts with the radio. I'll have to install the tuning pot and RF gain controls which will have to hang off the board for now.



Audio amplifier and active filter installed, the analog switch can be seen in upper left.

Archive for December, 2009

Call For “First Builders”

December 10th, 2009

It's a busy time of the year, Christmas is fast approaching. I have been tweaking the openQRP rig here and there and making contacts when I have time. It is working well and I think it's time to take the next step and see if there are any folks interested in being first builders. I am looking at offering 20 kits, to be sent out toward the middle of January. A kit will consist of all components including an unpainted chassis base plate, LCD, and controls. I have to determine kit cost, it's looking like something around \$120. Shipping cost will be \$5 for US. Click on the link in the sidebar to get a list of the specifications. The top cover of the rig has not been fabricated yet and won't be available until April. I will send one out to all builders as soon as I get them.

The only extra thing you'll need, besides standard kit assembly tools, will be the USBtiny programmer. This is required to upload the radio's firmware. I just have a bare bones code base at the moment but this will improve as we move along. If you have an interest in contributing to the firmware development, that will surely be welcomed. There is lots of room in the CPU's code memory for some nice user interface features. Also I am open to mods and design suggestions.

Since this is a first build, the assembly documentation will be brand new, so it certainly will be a job for experienced builders. I am planning to use the forum for questions and answers and I will be adding more content to the blog in the next couple of weeks. The only test equipment required will be a good multimeter and a calibrated receiver. A low power wattmeter is helpful, and of course a good 40 meter antenna is essential.

So if you have any interest or questions drop me an email using the 'Contact Us' link on the bottom of the page and I will start to assemble a list.

73 Steve

Archive for January, 2010

Project Update

January 18th, 2010

As promised in the last post, I measured the transmitter output with a spectrum analyzer and found some unacceptable spurs that prevented the design from meeting FCC guidelines. I spent the past two weeks attempting to fix the problem without resorting to another PCB revision. Sad to say, it is not possible to adequately fix the problem without changing the PCB layout. I have made the changes and am currently doing final checks before sending out for new boards.

I will post a discussion of the problem along with spectrum analyzer pictures after the PCB changes are complete.

Needless to say this will delay the first run kits.

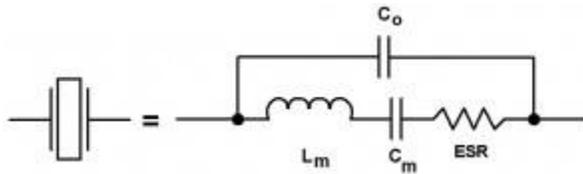
Crystal Filter Characterization

January 8th, 2010

Lattice Filter Design

Several years ago a friend of mine gave me several thousand 4.9152 MHz crystals. Originally these crystals were intended to be used as microprocessor clocks, (if you divide this frequency down, you get standard serial baud rates: $4,915,200/512=9600$) Not knowing anything about these crystals, I thought it would be a good idea to characterize them and see if they could be used for a crystal lattice filter. I followed procedures presented in Wes Hayward's book *Experimental Methods in RF Design*. (EMRFD) This involves looking at the individual

crystals in the lattice as complex networks of inductance, capacitance, and resistance and then considering all of these elements in the design of the filter. As shown in the diagram below there are four main components to a crystal:



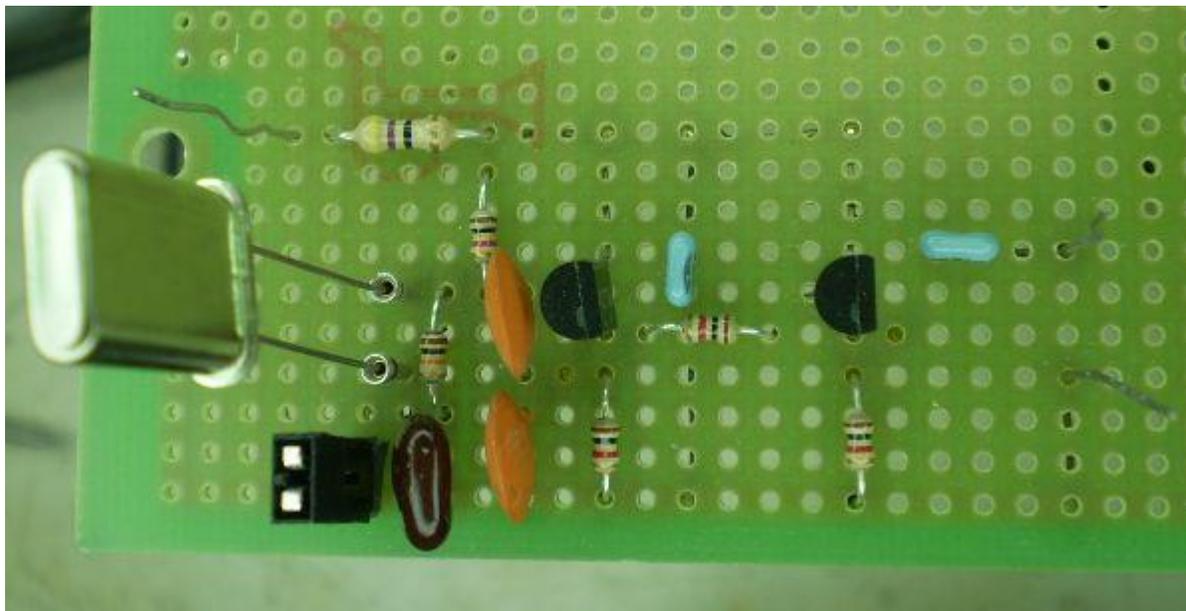
C_o = Parallel capacitance of the crystal holder plates

C_m = Motional Capacitance

L_m = Motional Inductance

ESR = Effective series resistance

There are several ways to determine these values, one fairly easy way, which is referred to in EMRFD, is the G3UUR Shifted frequency method. It's not the most accurate means but it's good enough for what I need to do. The way it works is the candidate crystal is installed in a Colpitts oscillator circuit that allows a fixed small capacitance to be switched across the crystal. By knowing the value of this small capacitance and the frequency shift it causes, both L_m and C_m can be calculated.



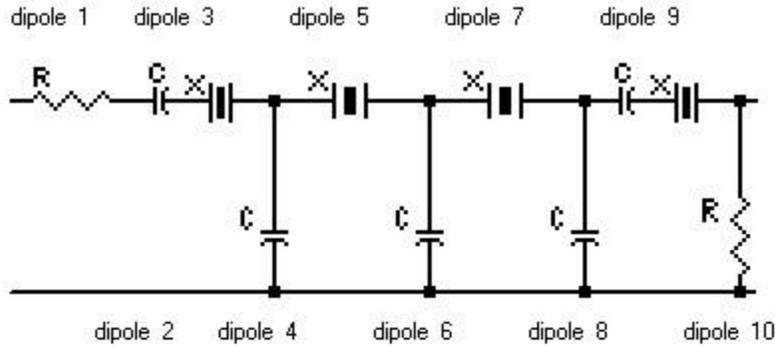
I built the circuit illustrated in EMRFD Fig. 7.69 (shown above) and collected some test data using several different crystals. I have an impedance bridge which allowed me to directly measure the value of C_o which was 4.6 pF. I also was able to measure the series resistance and found that to be about 38 ohms. I accurately measured the shift capacitor to be 51pF. The average measured shift with this capacitor was 671 Hz.

Plugging these values into the G3UUR equations, I calculated

$$C_m = .01392 \text{ pF}$$

$L_m = 75.295 \text{ mH}$

Next step is to calculate the filter component values. I used the [AADE filter design tool](#) which is very powerful. I simply select the type of filter I want to design (Crystal Ladder) and then select Classic→Butterworth. Next I click on "Cp,Ls, Cs, Rs Known" and then I enter four poles with a bandwidth of 550 Hz. I select tuned Capacitive Coupling as the desired topology. After the application calculates the design values, a schematic of the filter is presented.



DIPOLE 1
R 1=339.969

DIPOLE 3
C 3=229.90019pF
Crystal
 $F_s=4.916063\text{MHz}$
 $F_p=4.923496\text{MHz}$

DIPOLE 4
C 4=147.96244pF

DIPOLE 5
Crystal
 $F_s=4.916063\text{MHz}$
 $F_p=4.923496\text{MHz}$

DIPOLE 6
C 6=229.90019pF

DIPOLE 7
Crystal
 $F_s=4.916063\text{MHz}$
 $F_p=4.923496\text{MHz}$

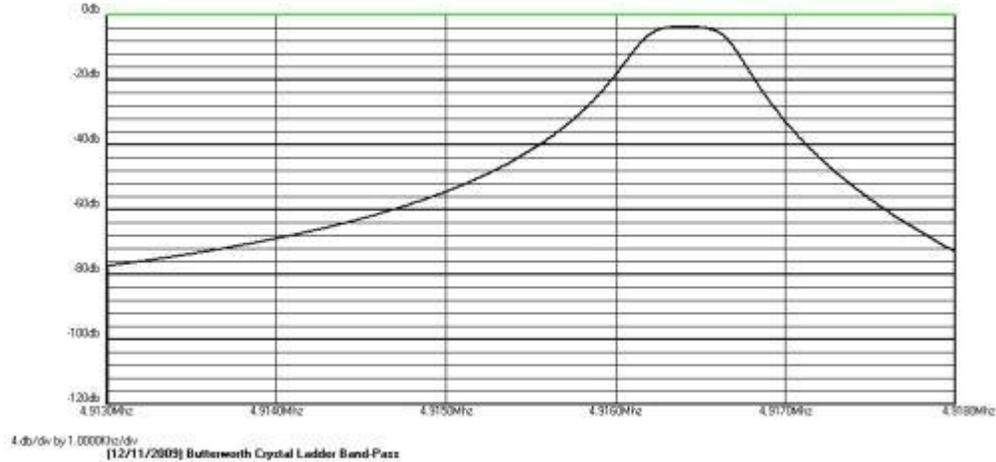
DIPOLE 8
C 8=147.96244pF

DIPOLE 9
C 9=229.90019pF
Crystal
 $F_s=4.916063\text{MHz}$
 $F_p=4.923496\text{MHz}$

DIPOLE 10
R 10=339.969

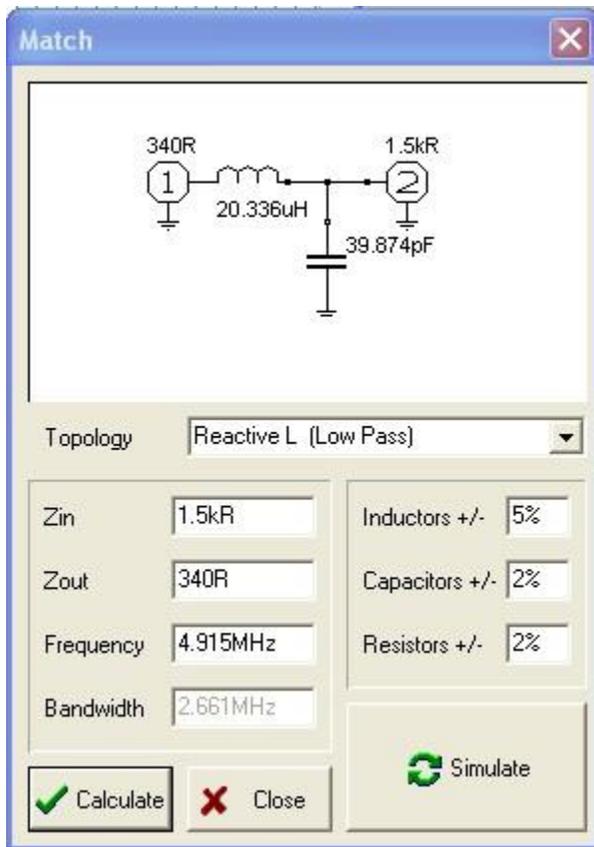
4.th order (12/11/2009) Butterworth Crystal Ladder Band-Pass
Bandwidth = 550.hz @ 3 db
Design Impedance=339.969 ohms
Input Impedance = 339.969 ohms
Output Impedance = 339.969 ohms

I am also able to plot filter response to get an idea of what it should look like.

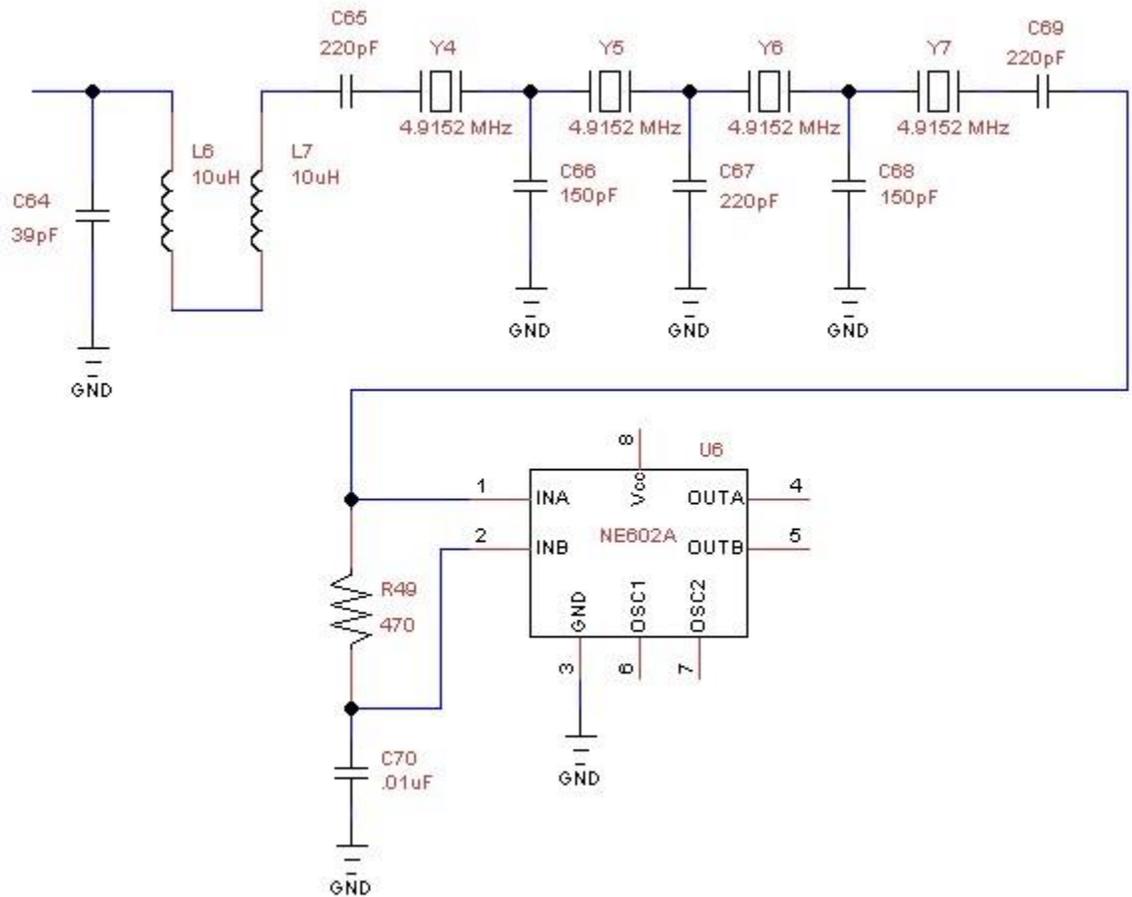


Filter Input and Output Impedance Matching

Now that I know the input and output impedance of the filter I can look at matching the filter to my receiver design. The filter input impedance is 340 ohms, the output impedance of the NE602 mixer is 1500 ohms. I will use an LC impedance matching network to join the two. A good tool for this is [RFSIM99](#), it has many useful functions, and the one I use here is the LC impedance match calculator. I select Tools→Design→Match and enter the values in reverse since the impedance transform is reciprocal. I get a value of 20.336 uH series inductance and 39.874 pF shunt capacitance.



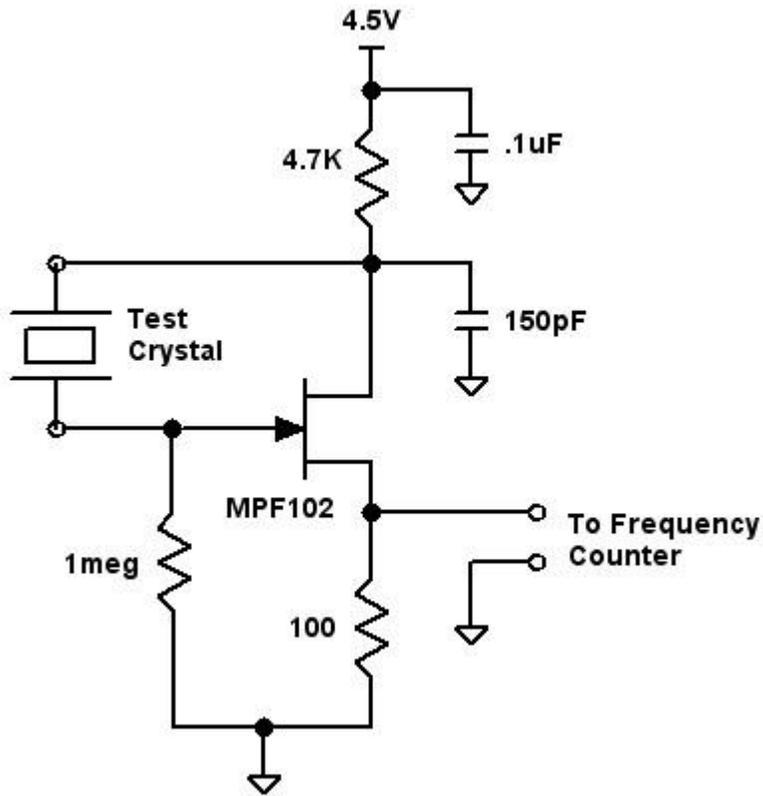
I rounded to standard values, 20 uH and 39pF. For some reason it is difficult to find 20 uH inductors so I chose to design the board to accept two 10 uH inductors. This works out well from the standpoint that there are other 10 uH inductors in the design already. The output impedance of the filter is 340 ohms, since the input impedance of the NE602 is 1500 ohms I terminate the filter with a 470 ohm resistor, this in parallel with the NE602's 1500 ohm input gives a termination value of 350 ohms.



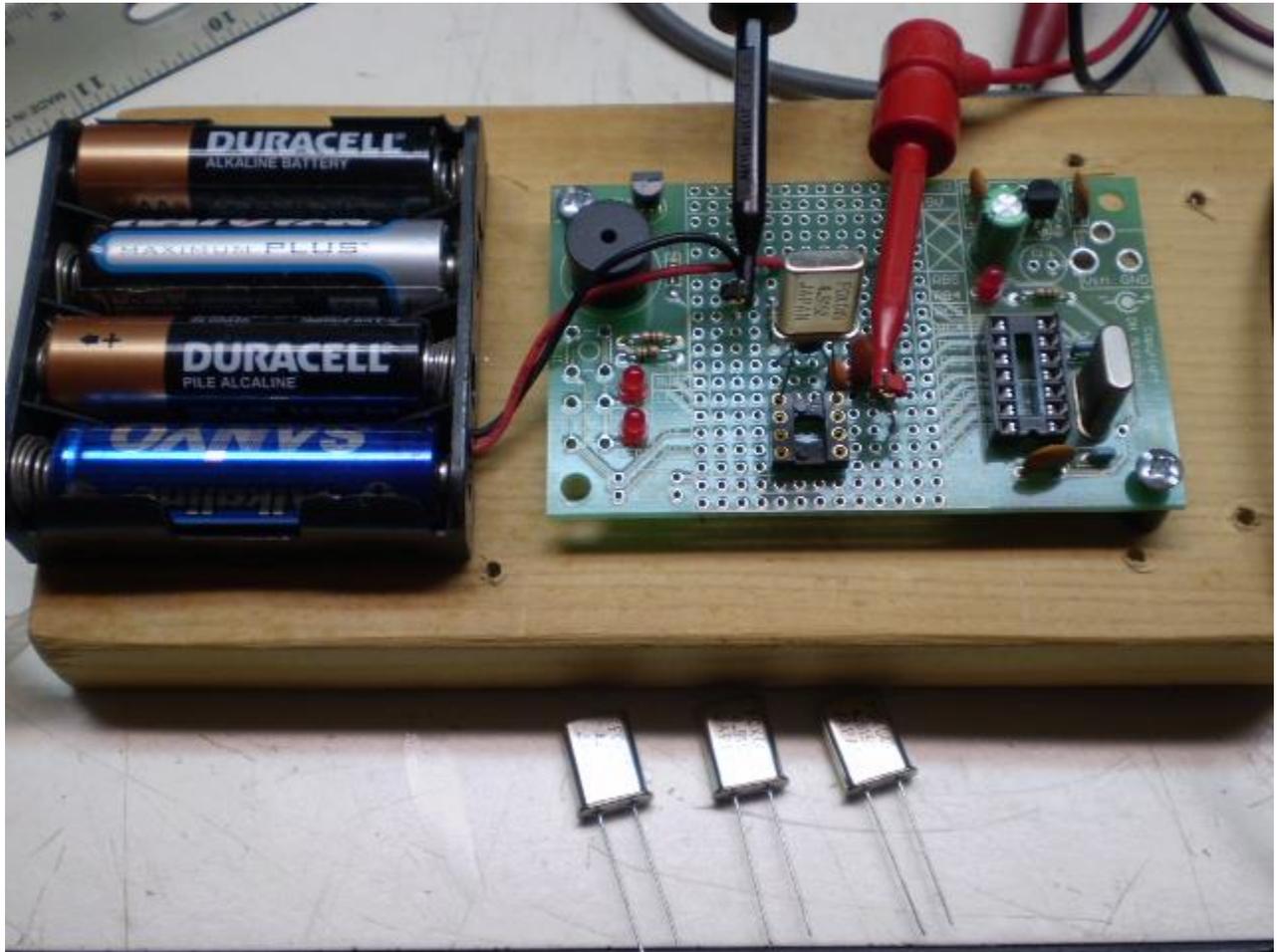
Now that I have component values for the filter and the input and output impedance matching figured out I am very close to being finished.

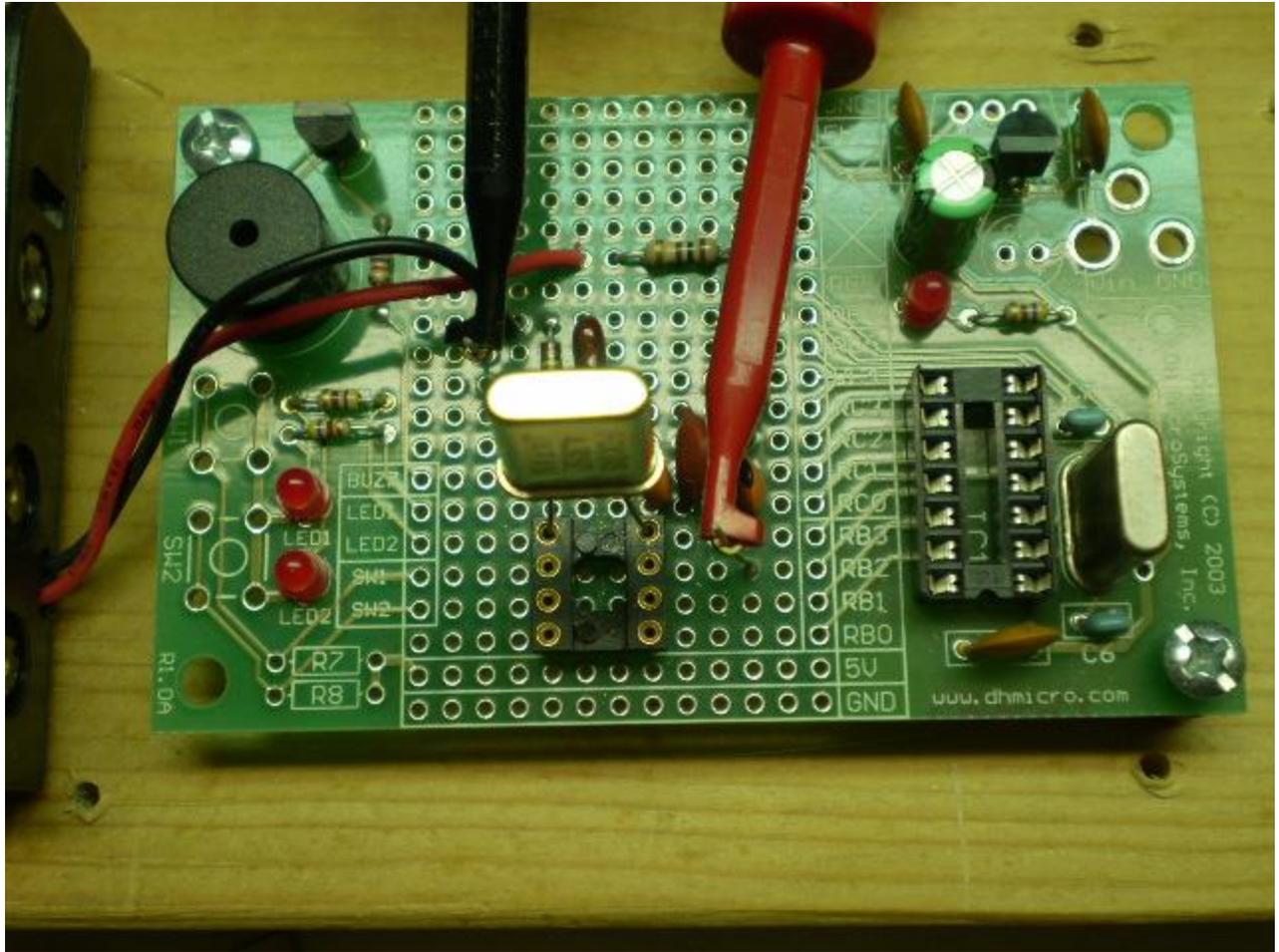
Crystal Matching

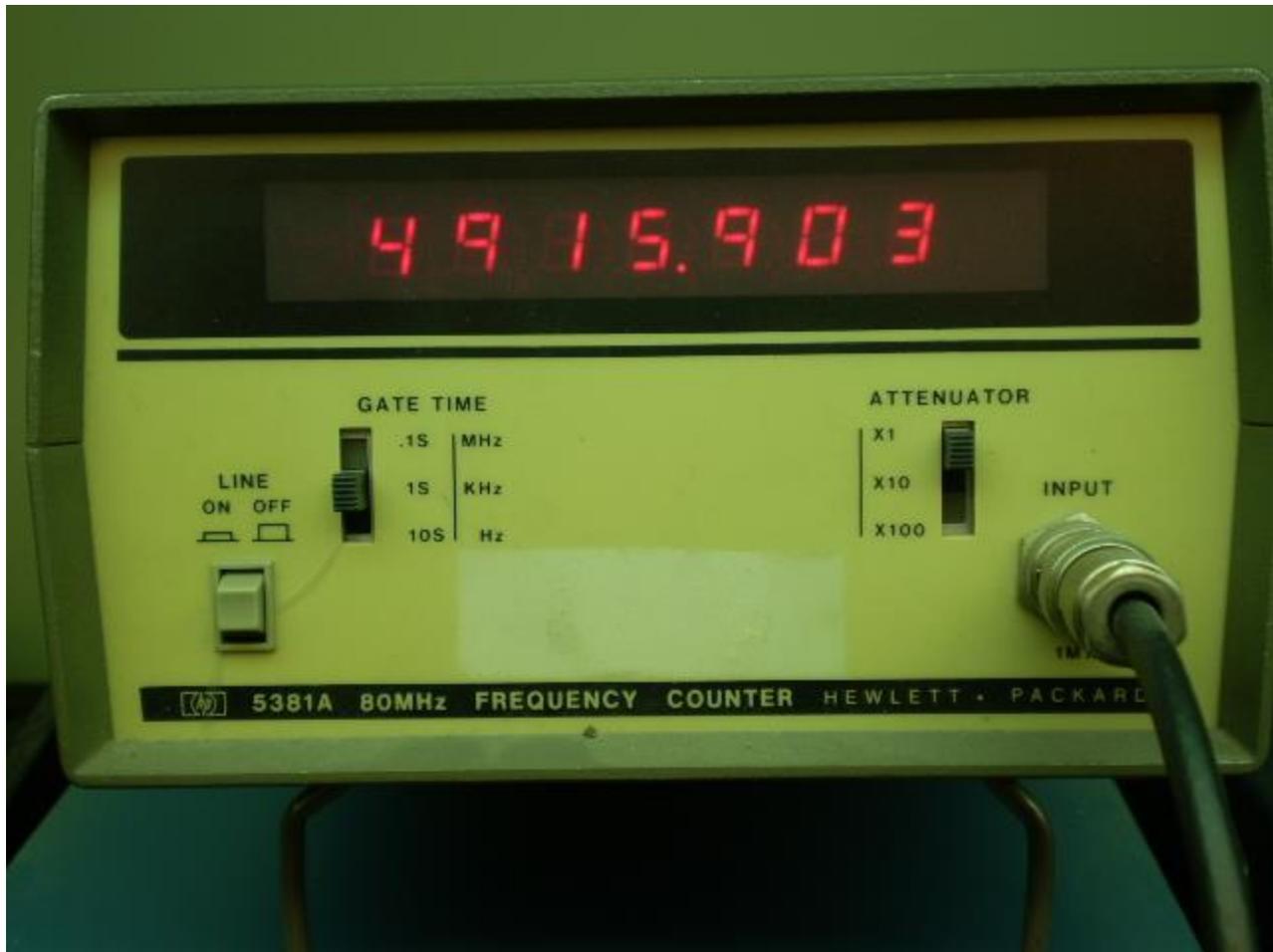
The next task is to match crystals for the filter. I followed the rule of thumb which states that the frequency spread of the crystals in the filter should be within 10% of the filter's bandwidth, this is 55 Hz. The crystal batch I have to choose from are reasonably close in value such that it was very easy to match sets to that tolerance, actually I was able to do better than this. I used the following oscillator circuit connected to a frequency counter to screen values. I use a length of Styrofoam and divided it up into 20 sections. I then checked each crystal and placed it into the appropriate section. The divisions were in 20 Hz ranges starting at 4,915,500 through 4,916,000. After going through about 500 crystals I had a very good selection of matched crystals.



Following is a picture of my crystal oscillator screening board. It's a microprocessor prototype board left over from a previous project. The oscillator is constructed in the middle of the board. At some point I may install my crystal matching PIC design in the 14 pin socket and drive a dedicated LCD frequency display but for now my HP frequency counter works well enough.



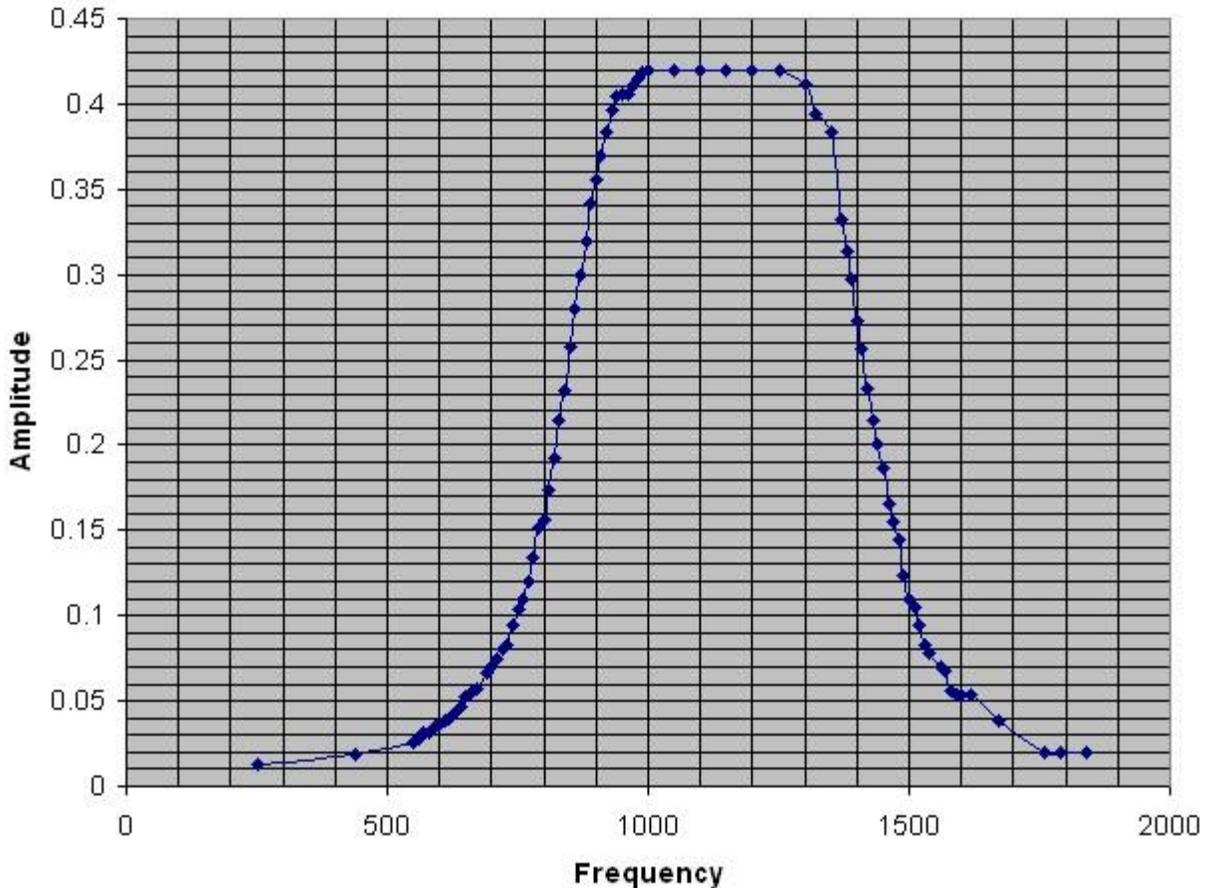




Filter Verification

Once the filter was installed in the receiver and everything was working, it was time to actually verify if the actual filter performance meets the predicted calculations. To do this I injected an accurate RF source into the input of the receiver and plotted the output level of the filter vs. frequency. I probed at the output of the second NE602 which insures that my scope probe capacitance doesn't become part of the filter and influence the measurements.

I set my RF generator to 7.020 MHz and tuned the signal in on the transceiver. I set the starting point just outside the filter passband and then manually adjusted the RF generator in 10 Hz steps and measured the output amplitude at each step. Note that I tuned the receiver's input bandpass filters to span about 4 times the measured frequency span, this insures that the bandpass filters minimally influence the crystal filter measurements. I recorded the levels in a spread sheet normalized the values and plotted the results:



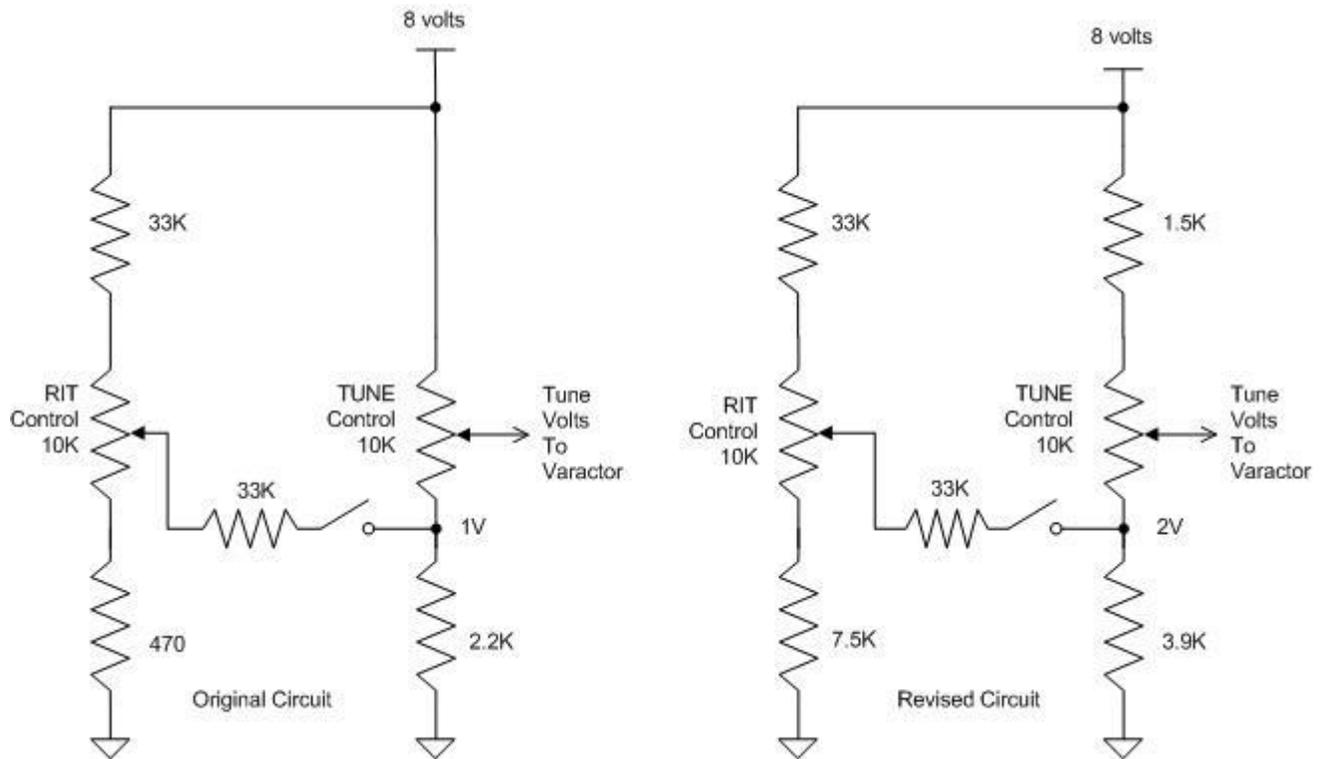
The peak response is very flat and the -3dB points are at 550 Hz, exactly in line with the design specifications. Some may say that 550 Hz is too broad for CW work but I find it is actually very good, there is no ringing and the sharp skirts of the filter work very well for attenuating out of band signals. Additional selectivity is gained by the two stage active filter which follows the filter.

Happy New Year ! RIT/Tune Tweaks

January 4th, 2010

I have been doing some clean up on the remaining issues with the transceiver. One problem that I finally resolved was with tuning and RIT range. The tuning range was much wider than I had designed for, over 100 KHz, and the first quarter turn of the tuning pot barely moved the VFO frequency. The center point of the RIT was skewed to almost 75% of the RIT pot so that I did not have balanced plus and minus offset.

I went back to the schematic and measured voltages and figured out the problems. Shown below are two circuits, the one on the left is the original design, the left is the improved version.



To analyze this circuit look at the RIT and TUNE sections separately. The values in the original RIT circuit set the RIT voltage at 1.0V with the RIT pot at midrange. The values in the TUNE circuit set the voltage at the bottom of the TUNE pot to 1.0V. When the RIT is set at midrange and the RIT switch is closed, there is no current flow between the TUNE section and the RIT section. If the RIT control is set to provide a higher voltage than 1.0V, then current will flow into the TUNE circuit increasing the voltage across the bottom resistor which increases the tuning voltage. When the RIT control is adjusted to provide a voltage less than one volt, current will flow from the tune circuit into the RIT circuit decreasing the voltage drop on the bottom tune resistor which lowers the Tune voltage. This gives us two nice features, the first is that the RIT offset is independent of the TUNE pot position which means an equal RIT voltage swing is possible across the entire range of the tuning pot. The second feature is that the maximum voltage applied to the analog switch is approx. 2 volts which is less than the maximum allowable input voltage to the analog switch which is powered off of the 5 volt supply.

In the improved version, the null RIT voltage is raised to 2 volts. This requires that the minimum tuning voltage is increased to 2 volts also. I found that varactor capacitance change was minimal in the range between 1V and 2V and this accounted for the dead zone in the tuning pot. Raising the minimum voltage to 2 volts provides a steady change in capacitance vs. voltage throughout the tuning pot range. The next problem addressed was that RIT adjustment was not possible when the Tune pot was set to maximum, this is because the tuning voltage was sitting at the 8 volt rail and the RIT circuit was not able to move the tuning voltage in either direction. Inserting a resistance between the top of the tuning pot and 8 volts allows the RIT offset to work at all settings of the Tune pot. Both of these fixes address a third

problem which was a very compressed tuning range on the Tune pot. Eliminating the dead zone brought back a 1/4 turn of tuning control and lowering the upper Tune voltage reduced the overall tuning range to around 80 KHz, still plenty of room for the 40M CW band. I might drop the range down to 70 KHz for an even nicer tuning ratio.

The skewed RIT range was due to the fact I populated an incorrect resistor in the RIT circuit, with correct values the null RIT point is directly in the middle of the RIT pot.

So technically this gives us a PC board change, but inserting a resistor in the high leg of the tuning pot is fairly easy to do and I think I will leave it at that for now.

I have started kitting 20 beta kits, and have had a good response to the call for builders. I have made corrections to the mechanicals for the chassis base plates, I should be able to get new ones turned around in about two to three weeks. I have found a few parts shortages for the beta kits but will be able to get them easily.

The only other task I want to complete is Tx spectral purity testing, I will work on that this week and report the results.

Archive for February, 2010

REV C Board Picture

February 10th, 2010

Can you spot the changes ? There are quite a few. So far CPU and display are running.



The obvious ones are the dual IF cans after the 1st Tx mixer and the new AF amplifier. Also there is only one TS922 op amp now. To make room for the 2nd IF can I had to move many components in the upper left. The LCD contrast control is now a smaller one right behind the display.

Rev C Boards

February 8th, 2010

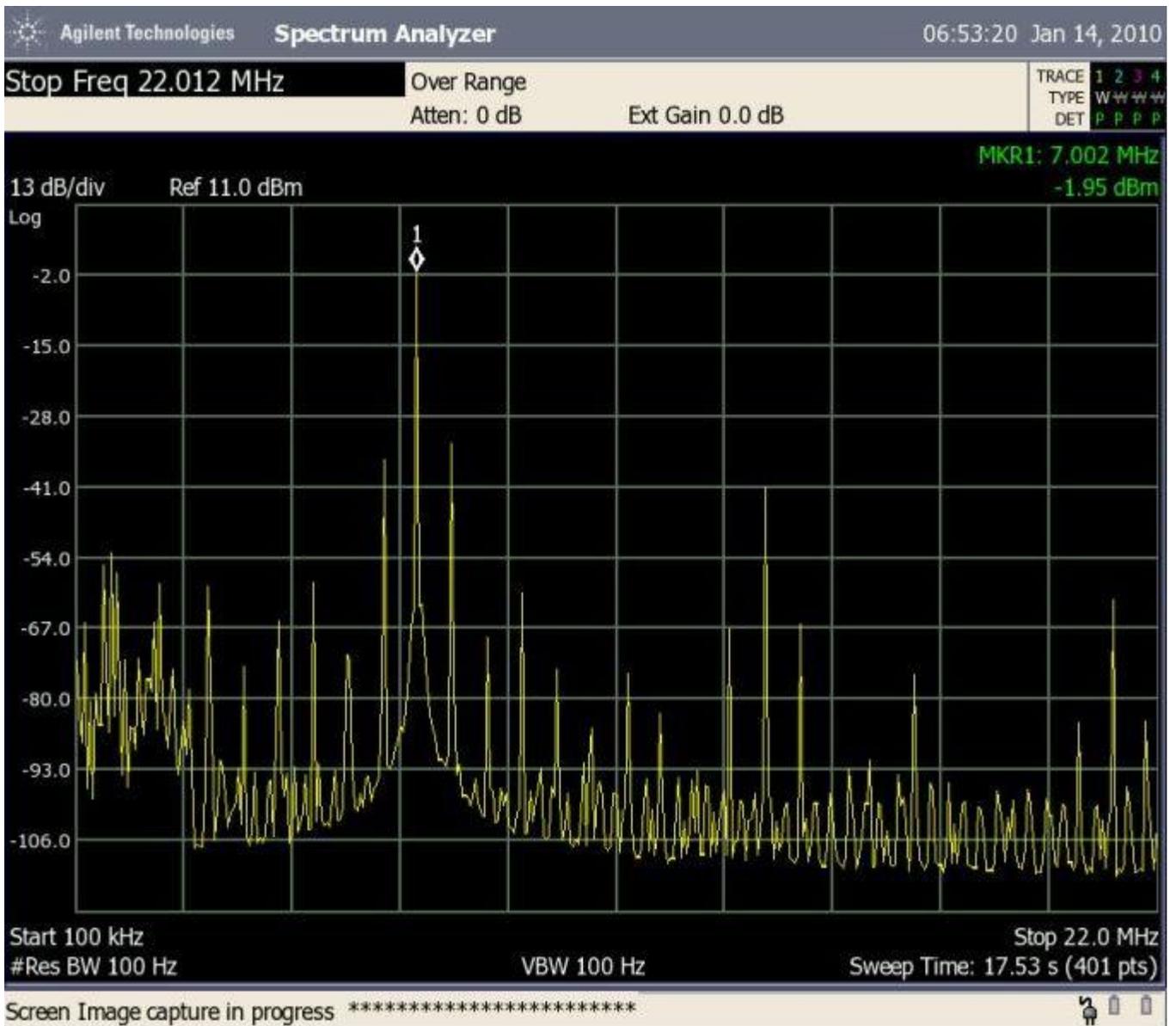
I received the Rev C boards today. Also will pick up the updated chassis metal work today.

A busy week ahead.

Fun With Spectral Analysis

February 3rd, 2010

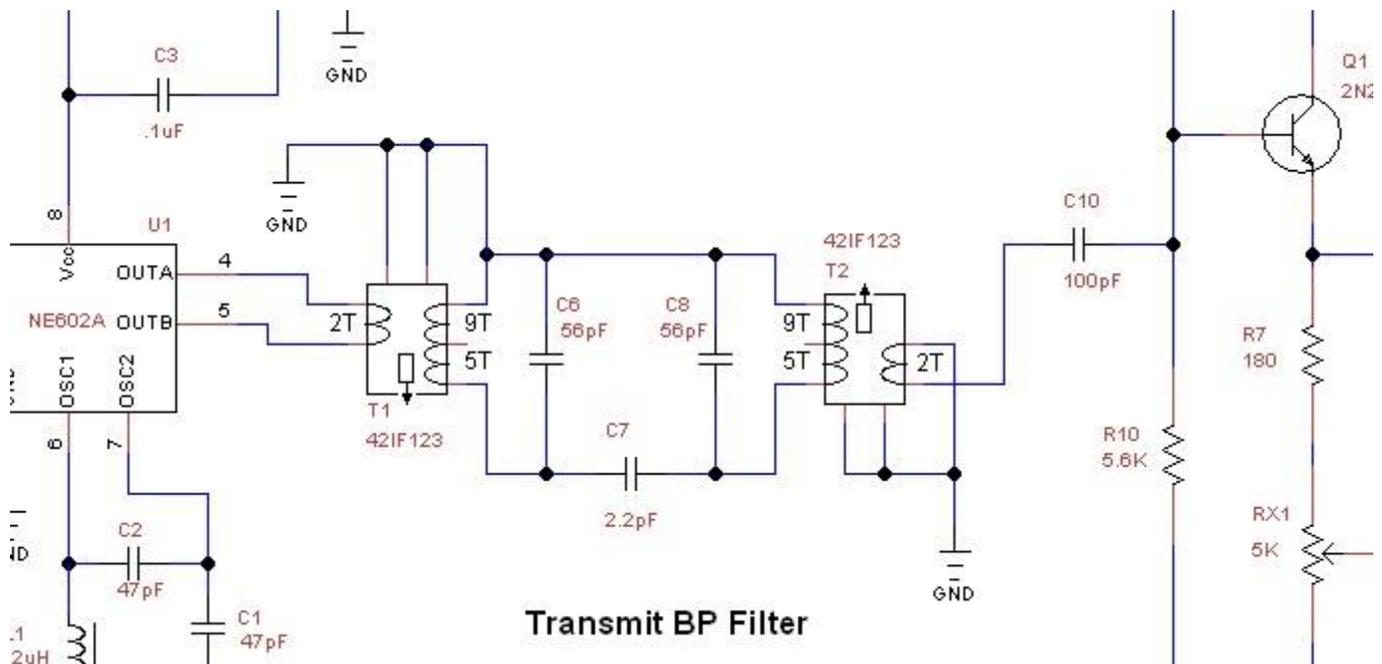
Back on October 29 of last year, I posted a description of the OQ transmitter with a few pictures of what I thought were clean transmit output waveforms. Indeed the output waveform did look very sinusoidal and I thought I was done with the transmitter design. I had an opportunity to look at the output with an Agilent N1996A Spectrum Analyzer, I was surprised at what I found.



Besides the fact that there were a sea of spurs, there were two peaks on either side of the fundamental that looked very much like modulation sidebands. The fundamental was 7.002 MHz with the low side spur, 6.26 MHz, at -36 dB and the high side spur, 7.743 MHz, at -32 dB. These were related to the fundamental in that they repeated with each harmonic, but unlike modulation sidebands, they were not symmetric. My first thought was that I had some sort of parasitic oscillation but really could not find anything like that. I was very grateful for some pointers from Steve Weber KD1JV and Jason Mildrum NT7S who both figured they were not sidebands but actually due to 2nd and or 3rd harmonic effects in the transmit mixer. If you take the mixer crystal frequency of 4.9152 MHz, multiply it by two, and mix it with the LO you get $(4.9152 \times 2) - 2.086 \text{ MHz} = 7.744 \text{ MHz}$, that's one spur. The other spur is simply the third harmonic of the LO ($2.086 \times 3 = 6.28 \text{ MHz}$). The main design problem was not having adequate filtering following the TX mixer, so undesired mixer products were not attenuated adequately.

Improved Tx Mixer Filtering

The reasonable solution was to add a second filter stage after the Tx mixer. The following schematic shows the new Tx mixer filter. I now use two IF cans and connect the low impedance side of the first one to the output of the NE602. The tunable LC tanks circuits of the two IF cans are loosely coupled and the low impedance winding of the second can is connected to the first driver input. This is a very effective filter with the down side that the tuning is quite sharp and the two cans have to be staggered tuned to get a reasonably wide response to cover the 40 M CW band. A better solution would have been to use a higher LO frequency, say 12 MHz, which would have moved the harmonic responses well outside a single stage BP filter passband. That would have meant a redesign of the VFO, receiver, and crystal filter. More work than I really have time for right now.



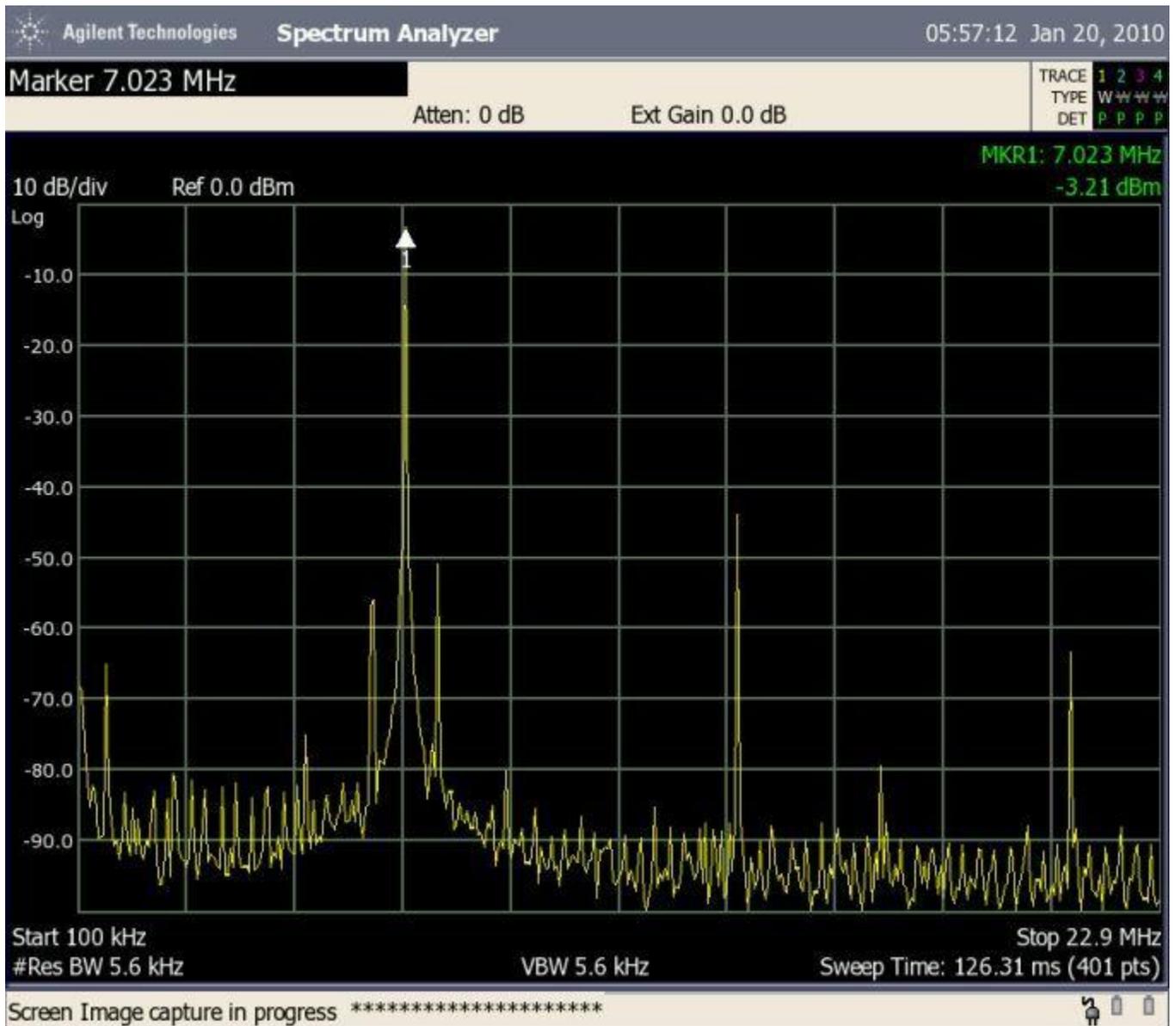
Improved Tx Mixer LO

Since I had to make modifications to the board for the additional filter stage I also decided to tap the Tx mixer LO off of the VFO tank circuit to get the cleanest signal possible. In the original design, I buffered the VFO before feeding it to the 1st Rx mixer and the Tx mixer and this caused some distortion of the LO waveform. I tried re-biasing the amplifier and adjusting the gain but really could not come up with a satisfactory result so I switched to a direct feed.

Second SA Run

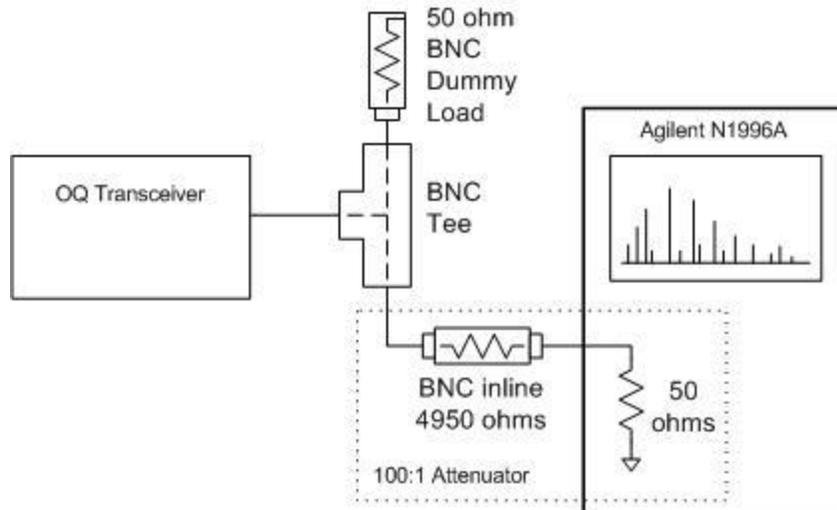
Following is the spectrum analyzer display illustrating the net effect of a clean LO and improved Tx mixer filtering. Notice that the sidebands are still there but down more than 50 dB. Also notice that the other numerous spurs are now way down, the original mixer design generated lots of extraneous stuff that was making its way through the single stage filter. The 2nd

harmonic at 14 MHz is down just over 40 dB and the third harmonic is down over 60 dB. The "sideband" spurs are down over 50 dB. The transmitter design now meet FCC requirements.



Spectrum Analyzer Setup

As a side note, I have not really found much information on the net or in books covering how to connect a Spectrum Analyzer to a transmitter to examine spectral content. Spectrum Analyzers generally have a 50 ohm input impedance but most are not capable of accepting 5 watts of power directly into the input port. The signal must be attenuated in such a way that a 50 ohm load impedance is maintained on the transmitter. The optimal way is to use a directional coupler or RF tap neither of which I had. So I devised the following setup.



Using easily found components, this RF attenuator provides a 100:1 reduction in signal amplitude and is well shielded. A 55 volt peak to peak transmitter output is dropped down to .55 V or -.97 dBm RMS into 50 ohms. Shielding is very important to preventing extraneous signals from other sources appearing in the sweep. The load on the transmitter is lowered slightly to 49.5 ohms but it remains resistive and does not alter the output significantly. The 4950 ohm part of the divider is a 50 ohm BNC inline terminator that I took apart and replaced the 50 ohm resistor with a 4950 ohm. The key to this setup is the use of the 50 ohm input impedance of the spectrum analyzer as part of the attenuation network.

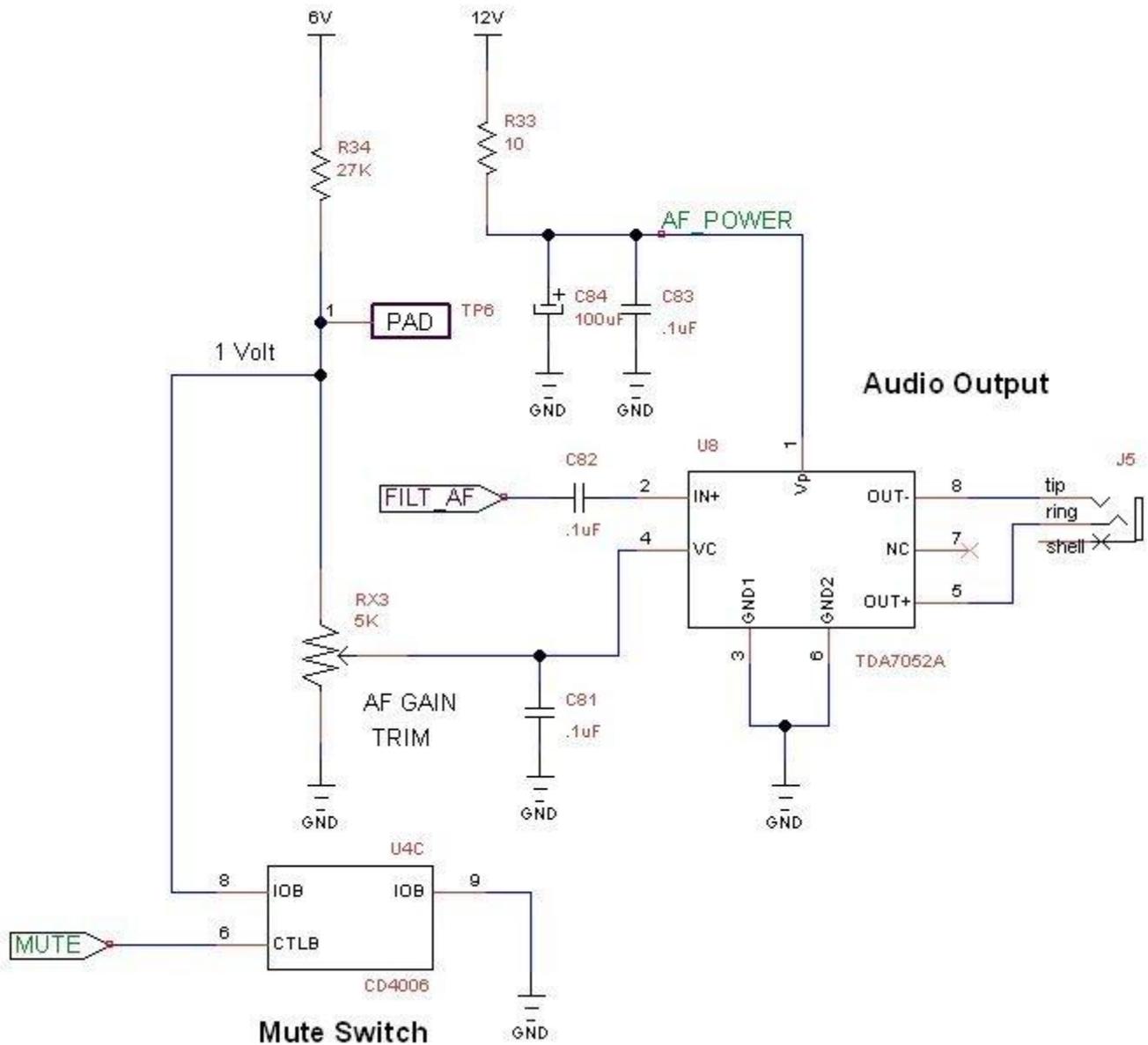
New Board Layout



So what does all this mean for the PC board layout ? As you can tell from the picture, the changes are quite messy and not easy to duplicate by the builder so that meant a new board revision. Officially, the OQ PCB is now at REV C. While I was making the transmit changes, I decided to address several other issues that I have been having trouble with.

Pops and Clicks

Despite my best efforts, I have been plagued with audio glitches during receive/transmit changeover. The analog switch muting worked well but not in all cases. Occasionally I would get pretty loud clicks during paddle operation especially if I ran in fast QSK mode. No reasonable amount of filtering could completely fix it, I won't get into the details here, the analog switch idea just had too many flaws. I decided to remove the second op-amp audio stage and replace it with a TDA7052A single IC audio amplifier. The original dual stage active filter isn't necessary, one stage is just fine and I am now able to drive a speaker. I really like the '7052, it has a BTL output which runs in a push pull mode and it has variable gain control via DC voltage level. This allows a real nice way of muting the receiver by simply pulling the gain control line to ground. This also provides an easy way to add AGC in the future.



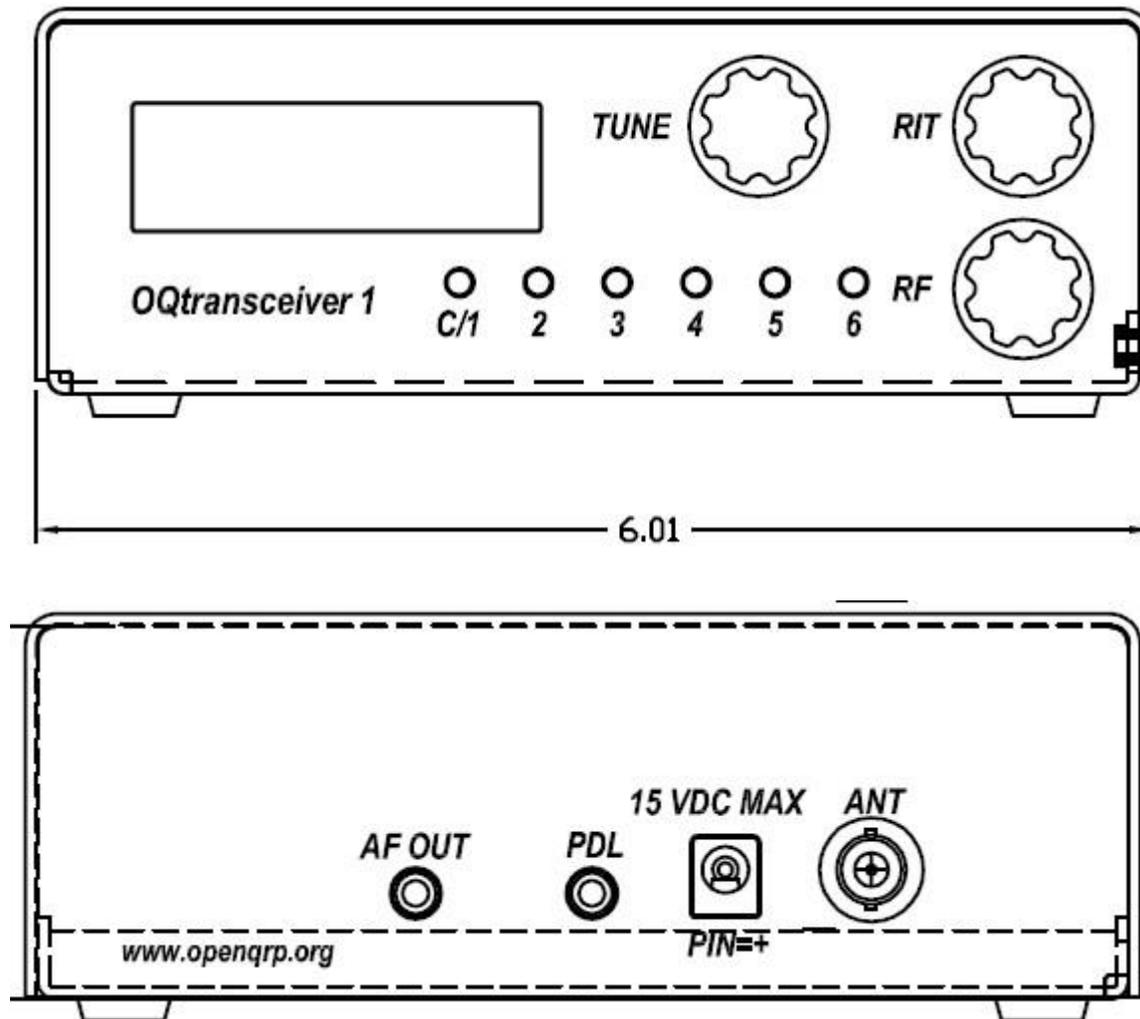
The new board is out for PCB fabrication and is due back on or around Feb. 15. REV C schematics, in PDF format, are posted on the right sidebar while the schematic source and PCB layout files are posted in the Design Files/Tools section.

Archive for March, 2010

Silkscreen Wins

March 11th, 2010

Here's the final drawings of the OQ enclosure, it is black powder coated with white silkscreen.



Transceiver Update

March 3rd, 2010

Life in Hew England

I have been sidetracked with some vacation time, sick kids, and five days without power. (I think it's a requirement now to own a generator if you live in the Northeastern US) So... progress has been limited over the past two weeks. I have verified the new version PC board and it all works fine. The beta kits are about half done, there are a couple of parts that are on backorder from Mouser, IRF510s and a couple of resistor values. Through-hole parts still seem to be available although they are not stocked like they used to be. Many common parts and values are now special order with minimum quantity requirements. It doesn't matter much to me since I buy in large quantities but for the person who wants to buy a few parts for a project, it is getting more difficult.

Arduino Notes

I noticed the other day that Arduino has released a new software package which has better support for custom hardware. I need to download it and learn how that works. On a different note, I have had some trouble uploading sketches to the new revision xcvr board. The workaround is to externally power the board during programming. I have not figured out exactly what the difference is but I suspect that I just got lucky on the previous version board and that the current draw of the xcvr board is more than the USB powered programmer can handle. Officially, I will recommend using external power mode when using the USB Tiny programmer. To do this, the power jumper, which is close to the side where the ribbon cable attaches, must be removed. This disconnects USB Tiny power from the target board. I have had very good luck programming in this mode but it's slightly inconvenient since I have to connect power to the board before attaching USB Tiny.

Enclosure News

I have decided to include a full enclosure with the beta kits. Originally I was going to include the enclosure bottom only. The enclosure will also be powder coated so it'll be a nice finished product. The only thing I am on the fence about is whether to add a silkscreen legend since it adds about \$10 to the cost. Maybe folks can let me know whether it's worth that to have silkscreening on the front and back of the enclosure.

Multi-Turn Tuning Pot

I have located a source for a reasonably priced multi-turn potentiometer and this will be included in the kit as well. It's a big improvement over the standard pot and makes fine tuning much easier.

Archive for May, 2010

Project Update

May 26th, 2010

100 radio enclosures are due the end of this week. Getting the design correct has been a challenge. As soon as they come in and I verify that everything fits together correctly I will be able to relax. I will post a picture as soon as I get them.

Archive for June, 2010

Crystal Matching

June 30th, 2010

I am currently about halfway through matching crystals for the Rx Filter and Tx Mixer, it is going much easier than I thought it would. The crystals I have are very high quality and are very close in frequency. I did 14 sets last night and will do 14 more tonight.

Lots of progress has been made in kitting, all that is left is the receiver. I have been kitting the transceiver in sections, three or four sections per plastic bag. This will make assembly easier

since each section is a self-contained portion that can be tested after installation. For example, the first section is power supplies followed by the CPU section. Next is the VFO followed by the output filter and Rx/Tx antenna switch. From there on to the receiver and transmitter sections. This method makes kitting easier as well since there are only a handful of parts per section are easier to count and check.



Good news today, Digikey now has ATmega168 CPUs back in stock, this is the last part I need to order. For some reason these are difficult to find right now, maybe Arduino builders are buying them all ! Mouser has been out of stock for months and are still looking at 13 weeks before they get more.

Finally, an enclosure !

June 13th, 2010



Archive for July, 2010

Arduino Version 18

July 23rd, 2010

I am in the process of updating the CPU control firmware for the first release, I want to ship the beta kits with enough functionality to be able to run all of the features of the radio "out of the box". I expect folks will have updates and improvements as we go forward.

Anyhow, in the process of getting things ready I want to move to release 18 of the Arduino toolset. The main reason for this is that the notion of 3rd party platforms is now supported. What this means is that I can build a custom environment for the openQRP CPU which is self-contained and separate from the general Arduino release. In future when upgrading to newer Arduino releases, the openQRP specific files will not be affected. It has taken some time to figure out how it all works, but worth the effort since new openQRP code distributions will simply be one directory that includes all libraries, customized core files and header files. So to upgrade to a new openQRP version will just require the replacement of this one directory.

Beta Kit Interest (Anyone There) ?

July 9th, 2010

Considering all of the ups and downs over the past year, it's amazing that I am finally to the point of having beta kits ready to ship. On the other hand, it will be even more amazing if there is anyone left who hasn't given up on the whole thing and moved on.

So, in the interest of helping me figure out how many beta kits to ship, please drop me an [email](#) if you would like to participate in the beta kit build. I plan to email all those who have already "raised their hand" and I will not remove anyone from the list until I have gotten an ok to do so.

Thanks, Steve

Receiver Kitting

July 5th, 2010

On the home stretch now, I have the today off from work so we'll be kitting the two receiver sections, with some help from my daughter (9 yrs old)

Remaining tasks:

- 1) Package LCD display section and PSoC/AF Amp section.
- 2) Program Arduino CPUs & PSoCs, Package ICs
- 3) Package mechanical parts (knobs, heatsink bracket, screws)
- 4) Package controls/connectors (RIT, RF gain, 10 turn tuning pot, BNC)
- 5) Measure out copper wire for toroids
- 6) Box `em up

73 Steve

Archive for August, 2010 File Update Complete

August 24th, 2010

Reference and Design files are now updated and reposted.

Design File Updates

August 23rd, 2010

I am updating all of the design files, some of the ones that had been posted are out of date.

Some of the files will not be available for download today, but will be back on line tonight.

CW Keyer Module and Libraries

August 21st, 2010

A couple of years ago I exchanged some emails with avid contester Bob Wilson N6TV. He had one of my WKUSB keyer boxes and suggested that I add a particular keying emulation that he favored over others. WKUSB doesn't support exact keyer emulations, instead it allows you to tailor the paddle sampling mechanism which gets you pretty close.

Anyhow, to help illustrate how this particular emulation worked he emailed me an entire keyer he had written in C for the PC platform many years ago. I was astonished at the complexity and detail of the module. Unfortunately I was not able to implement any of Bob's algorithms in my keyer, it was impossible to translate it to PIC assembly in a way it would fit into the small code space I had available.

I archived the module and figured I would get back to it. One day when I was working on the C code for openQRP, I remembered the keyer module and wondered if it was possible to use it. It only took some minor modifications to get it to compile under Arduino and then a bit of interface tweaking to actually get it to work right. That's the beauty of portable languages. It's truly a great keyer and has a nice feel to it, maybe even better than my keyers.

I contacted Bob to ask him if I could get his permission to use his module in openQRP. The good news is he has consented and the keyer will be in the first beta release ! It really is a nice addition to the project. I am packaging it now as an Arduino library with a well-documented interface.

Just a quick brief on libraries, Arduino supports the notion of object oriented programming which has many advantages. Essentially a software program is broken up into functional modules that are separate entities from the main program. Each object is written and worked on separately and "connected" to the main program at run time. In Arduino land, libraries are objects, I already have a library that manages the LCD display and soon there will be a keyer library. I would like to have a separate libraries for CW message management, configuration storage, and many other things.

More importantly, the only reasonable way to foster multiple contributors to openQRP firmware is through the use of libraries which can be worked on separately and then brought into the main code base for integration and debug.

Another nice plus of objects is that if someone doesn't like the keyer or the way the display works, they can write their own version and replace it. To try to do that in a non-

object oriented software project is very difficult. Objects generally have a long life, they can be transported to new platforms and applications.

So that's it for today, back to the lab.

Why is this project taking so long ?

August 20th, 2010

I ask myself this question every day and a series of critical emails I have received lately encouraged me to write this post. I know folks are anxious to see something happen to prove this is not just an idea without fruit.

I formally started this project in December 2008. At that time I had a working prototype, most of the parts for 100 kits were in house, and it was all coming together nicely. In March 2009 I was on my way to work when I got tail ended by a truck going 65 MPH. My car was totaled and I was out of work for 2 months. I got hurt pretty bad and openQRP had to take a back burner for a while.

I have been playing catch up ever since and with a family (wife and two young children), full time job, a [keyer kit business](#), and a Mom with dementia, my life has been chock full of stuff. I'm no different than anyone else, and I'm certainly not complaining, I'm blessed in so many ways.

I am trying to funnel as much time as I can to get the beta kits shipped, I really underestimated how long it would take to get everything done and I humbly apologize for that, I'm just not as fast as I used to be. It's really analogous to priming a pump. Once the kits are out, firmware development can kick into high gear, and work on the assembly manual can get started. (You will be very surprised to hear who has expressed interest in working on the manual !) There are some other surprises as well. The main reason I decided to continue this project was the fact that it could be a community effort and I would not have to do it all myself.

So the project is very much alive, it's a labor of love, it will all be good !

73 Steve K1EL

Arduino 18 working

August 19th, 2010

There has been a large amount of interest in the group in the past couple of days.

I have successfully created a custom openQRP object for Arduino 18. It took quite a while since the process is not documented and I had to figure it out by trial and error. As with most of my work with Arduino the fastest way to make progress is to find some examples and learn from them. It's a very modular setup now, one folder with all of the related openQRP files in one spot. There is no longer any need to hack any of the standard Arduino distribution files. Even better, when new versions of Arduino are released all need happen is update a file pointer to the openQRP directory. Very nice addition to the Arduino IDE.

So I am just finishing up a base firmware release to get folks going and then add more functions as we get going.

I will be sending out emails next week to all those who expressed interest in being beta builders and depending on response we'll see how many extra kits I have to distribute beyond that.

73

Steve K1EL

Archive for September, 2010

Started Beta Kit Assembly Forum

September 28th, 2010

I have been working on the initial step by step assembly procedure and today posted it in the forum. There are still a couple of sections that have to be completed. It's a starting point, builders will be encouraged to add to the forums when problems or suggestions arise.

New Keyer now integrated

September 7th, 2010

I have completed the task of putting the N6TV Keyer module into an Arduino library and have it merged in with the transceiver code base. It provides the following CW keyer modes:

Curtis iambic A and B emulation

Accukeyer emulation

Smartkeyer emulation

Bug mode

Autospace can be enabled optionally. Also supported is paddle swap. All the basics.

The first release transceiver feature set includes:

Five 64 byte non-volatile editable message slots

Easy CW speed adjust

RIT On/Off

Full break in CW

Morse entered on the paddles in transmit is shown on the top line of the LCD display

CW decoded from PSoC displayed in the same manner during receive. (This can be turned off if desired)

VFO frequency display

Relative signal strength display

Power supply voltage display

I think that's about it. Enough to efficiently run the radio with lots of room for additional features.

Archive for October, 2010

PSoC Firmware

October 15th, 2010

I have spent the last few days getting ready to post the PSoC firmware. The PSoC controller is responsible for several important tasks. The first is CW decoding; not only does the PSoC decode CW that is entered on the paddles by the operator, it can also decode received CW live off air. In addition, the PSoC provides a real time signal strength indication that is displayed on a bar graph on the openQRP transceiver LCD display. This is used to tune CW stations in for decoding and also provide a relative signal strength indication.

Most of my time was spent just cleaning up and commenting the source code so that others can understand it better. While doing that I found several ways to improve the CW decoder accuracy. It works quite well for a single 8 pin chip running at 24 MHz. I copied W1AW this evening 100% which I could never do before. It now can handle speeds in excess of 40 WPM. There is still some room for improvement as far as noise immunity goes but all in all I am very happy with and am anxious for others to try it out.

PSoC stands for Programmable System on a Chip which is an invention of Cypress Microsystems. These devices come in many sizes and varieties and are used in iPods, laptops, appliances... the list is endless. The CW decoder presented here is not a work developed solely by me. The idea came from an application note published by Cypress back in 2003 when the PSoC was relatively new. This [app note was written by Melchor Varela EA4FRB](#) and is worth reading to get a basic understanding of how the decoder works. I started with Melchor's work and used his tone decoder with minor modifications. I rewrote the Morse decoder section in C based on CW decoders that I had written previously. It's a pretty good decoder but certainly can be improved upon. I have only used about 1/4 of the PSoC's code store so there is lots of room left to add some intelligence to the CW decoding algorithm.

The entire design is contained in one project directory that can be located anywhere on your hard drive. I recently updated it to build under the latest version of [PSoC Designer](#) generously provided by Cypress Microsystems for free download. You'll have to register and supply an email address and other information. Before we go too far I must tell you that this is not an undertaking for everyone. The PSoC architecture is quite complex and requires a knowledge of both digital and analog electronics. In addition you will need a PSoC programmer to program PSoC parts. The only source for this is Cypress, the lowest cost one is around \$185 US. But the PSoC is worth learning, since there are so many possible applications for it.

As mentioned previously I used Melchor's tone decoder setup to extract energy at the center frequency of 690 Hz. This is done with a quadrature correlator. Some of the recovered audio from the receiver is fed into the PSoC IC where it passes through two bandpass filters that are tuned slightly skewed from each other at 660 and 720 Hz. This slightly reduces the ringing effects of two cascaded SCAF filters. After filtering, the signal is converted to digital format by an A to D converter producing a sample every 181 uSec. The sine and cosine of these samples are fed into the quadrature correlator that effectively looks at eight successive samples and determines if these samples meet the criteria of a signal at a frequency of 690 Hz. The closer

the signal is to 690 Hz the larger the amplitude output from the correlator. Next the signal is subjected to some pretty heavy hysteresis to filter out drop outs and noise spikes. At this point we will have a signal amplitude that increases in value when dits or dahs are present and drops otherwise. The signal is thresholded such that a value below a certain level is ignored while a signal above a certain level is deemed a valid in band signal. Next the intervals are timed and a running tally of interval lengths is determined by an averaging median filter. Finally, using a dit/dah length threshold, dits are separated from dahs and loaded into a shift register. Letters are output when a longer space between elements is detected. The resulting digital byte is then looked up in a table to convert the serially assembled Morse letter into ASCII. It's sent over to the Arduino CPU and then displayed on the LCD.

Some challenges the decoder faces

The quadrature correlator works well in quiet band conditions but noise and fading really give it a difficult time. While my hysteresis setup works pretty well, it does time shift the signal and leaves it vulnerable to distortion. A better way might be to filter the values out of the quadrature decode with a moving mark space threshold based on average signal level rather than a fixed value.

The next challenge is adapting to the wide range of sending styles ops use. Decoding W1AW or other machine generated CW is pretty easy, you know that a dah is three times the width of a dit while the space between dits and dahs are one dit time, unless it's a letter space (3dit times) or a word space (7 dit times). In the real world you encounter all kinds of stuff, letters run together or stretched out, customized dit/dah ratios, and it's the Wild West out there. There are ways to make a decoder that adapts to a person's sending style in real time by analyzing and recording the different intervals of interest and using them as rules for decoding letters. I have started to work this out but it gets complicated rather quickly.

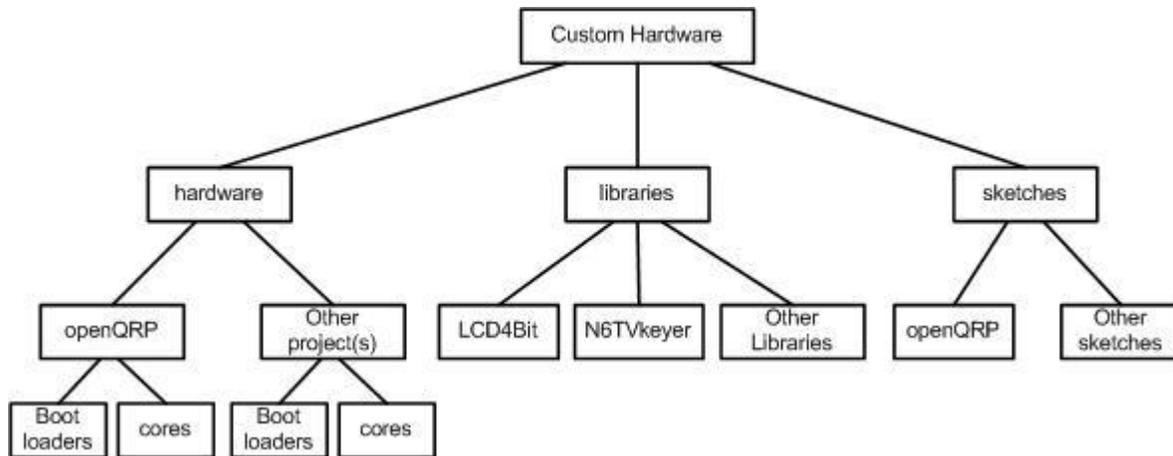
At any rate, a little more testing this weekend and I'll release the PSoC code and post it on Monday.

Initial OQ transceiver firmware release

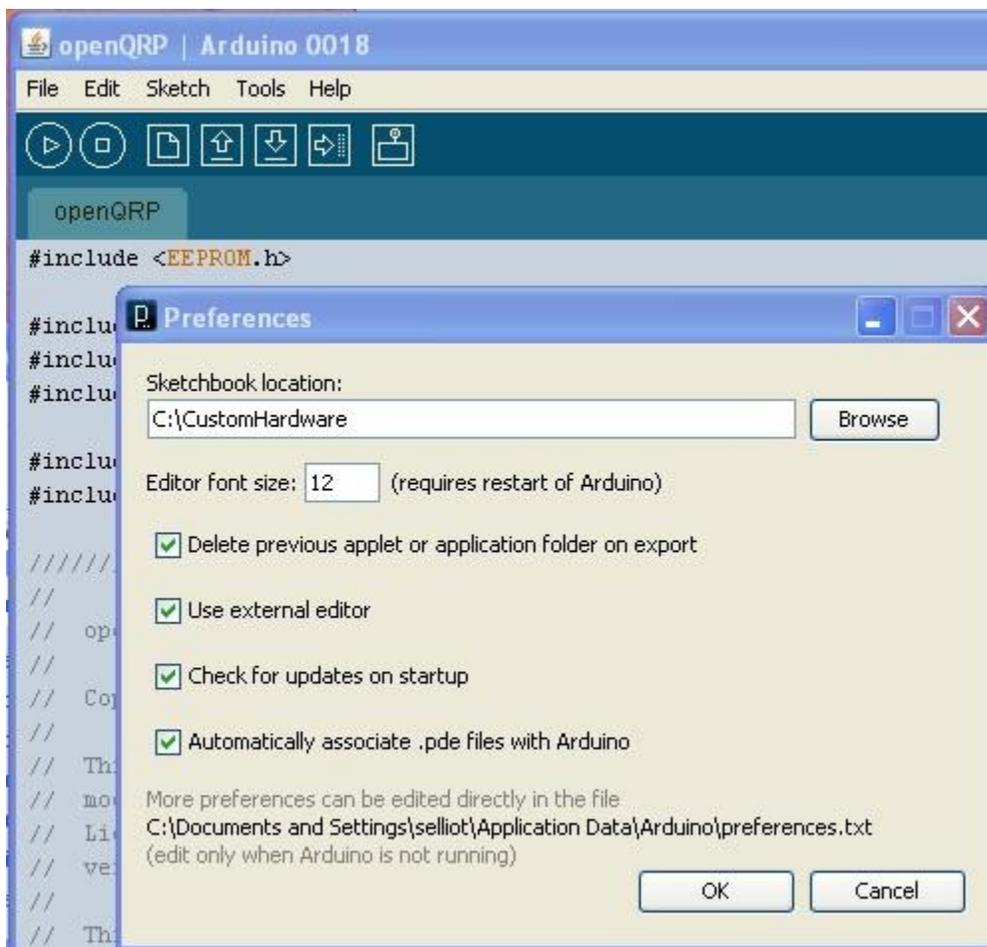
October 7th, 2010

Initial openQRP Application Release

This is the moment many have been waiting for. I am posting the Arduino firmware for the openQRP transceiver today. This was set up to work with Arduino release version 18 but will also work fine with their current release which is version 21. Starting with version 18 they made custom hardware development much easier. All of the files for a hardware project are now kept in a single directory outside of the standard Arduino installation. It's no longer necessary to hack files in the release core library or worry about overwriting your project when you upgrade Arduino. The structure of a typical CustomHardware directory looks like this:



As shown, custom projects, in addition to openQRP, can be added without too much trouble. I am assuming that most folks do not have a custom hardware directory set up yet so I am providing a pre-organized, zipped directory that you can extract to the root of your hard drive. (That's where I find it works the best, you may be able to figure out how to put it wherever you want) Once the directory is in place you need to tell Arduino about it, this is done by adding an entry to your Preferences dialog box:



Note that I have opted to use an external editor instead of the editor built into the Arduino IDE. I learned EMACs about 25 years ago when I was a software developer on the Sun Unix platform and I really don't like "windows based editors" but that's just me, there's nothing inherently wrong with the Arduino editor.

Now exit Arduino and restart the IDE to make sure the modified preference file has been obeyed, then select the openQRP sketch:

File->Sketchbook->sketches->openQRP

Make sure the correct hardware is selected:

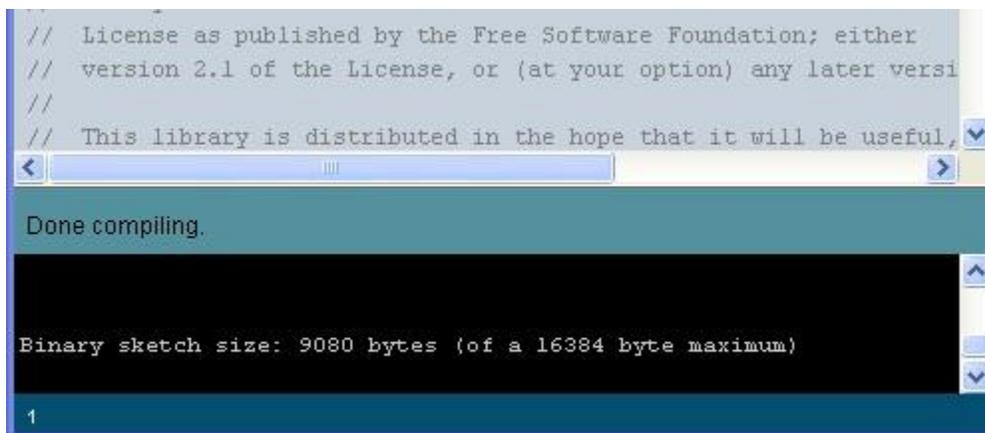
Tools->Boards->openQRP via USBtiny w/ATmega168

It is really cool to see openQRP appear as an Arduino built-in, it was not easy to figure out how to do this.

Now run a test compile:

Sketch->Verify/Compile

It chugs away for a while and then should display this:

A screenshot of the Arduino IDE's output window. The window has a light blue header with the text "Done compiling." Below this is a black area with white text that reads "Binary sketch size: 9080 bytes (of a 16384 byte maximum)". At the bottom of the window, there is a small blue box containing the number "1". The background of the IDE shows some code from a license file, including the text "License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version" and "This library is distributed in the hope that it will be useful,".

Just a few notes about the specific files that make up the project. There are really only a few files that you need to worry about, the first one is the sketch file:

/CustomHardware/sketches/openQRP/openQRP.pde. This is the source file for the main application that runs the transceiver, it's what you see when the Arduino IDE starts up. In this file you will find instantiations of one standard library: EEPROM, and two custom libraries: LCD4Bit and N6TVkeyer. These are located in /CustomHardware/libraries. I hacked one file that is part of the standard Arduino release, wiring.c, which can be found here:

/CustomHardware/hardware/openQRP/cores/openQRP/wiring.c

wiring.c hacks:

Specifically, I modified the Timer Zero interrupt routine to include the frequency counter capture code and also call the N6TVkeyer timing engine every millisecond via keyer_tick(). In addition, a function called frequency() was added to wiring.c to give the main app a way to obtain the

current frequency count. To make this all work properly, accompanying modifications were made to wiring.h. These are the only “hacks” to the Arduino standard release.

Closing Remarks

With this directory in hand you can now build the openQRP sketch and examine the basics of the control loop. The major giveaway here is the N6TV keyer. I know of no other freeware keyer out there that is as sophisticated or works as well as this one does. Since the keyer is run at interrupt level that means its timing is not disturbed by other functions. For example, the LCD interface requires significant delays to run properly. If the keyer was run as a function call from the main loop, you would hear distortions in keyer timing whenever the code went off to update the display while the keyer was running. We have the ability to do true multi-tasking with this thing.

You’ll find that the application just has basic functionality at this point. There is lots of code space left and lots of work left to be done.

In a few days I will post the source directory for the PSoC mixed signal controller. That will be a real adventure for those interested.

Getting ready to Post the Arduino and PSoC firmware files

October 6th, 2010

I worked on the Design Files tab today, just getting the sections and links in place. I have the Arduino portion ready to put up on the server, will do that tonight or tomorrow. The PSoC portion needs a bit more organization.

Archive for November, 2010

Beta Builders making progress

November 30th, 2010

Check out the

Step by Step Beta Build

Learn How to Assemble an Open QRP Transceiver

section of the forum, we are working our way through the first four kits and in the process, are putting together an accurate set of assembly instructions. Soon we will get back to the firmware and work on some improvements.

Forum back on line

November 15th, 2010

I refreshed all the forums settings, seems to be back to normal now.

Forum Meltdown

November 15th, 2010

Looks like we lost the forum today. I am working on getting it running again.

Kits !

November 10th, 2010

I have changed the beta kit plan slightly. Instead of sending our 25 beta kits, I have decided to send out just four kits. It just didn't make sense to try to manage 25 kits with minimal documentation and kit assembly instructions. Kits were sent on Monday, Nov 8, so we should start to see some action in the forum as things progress. As soon as we get through this build I will release the remainder of the 25 kits and continue with kitting the remaining 75 kits. Hopefully there will be enough interest to justify a **second kitting**.



Five kits, four to send one for reference.



Archive for December, 2010

Preliminary Assembly Guide

December 3rd, 2010

I have started a very preliminary assembly guide for the OQ transceiver. You can find it on the right hand sidebar under *Reference* along with the schematics and parts list. It will be refined as we finish up the beta tests. Many thanks to Ron W4DNQ for pictures and Don VE1AOE for the parts placement guide pictures.

Thanks to all the builders, especially Jason NT7S, for feedback, suggestions, and bug reports. Dean N7XG has been sidelined with a bad flu but he is back in action now.

How the kit is organized, by functional section.

Archive for January, 2011

Updated Assembly Guide

January 12th, 2011

I posted an updated OQ assembly guide that has some transmitter waveforms. Missing one key one, the gate of the final Q5. In addition I added considerably more detail in testing the transmitter stages.

Please note that I also posted some additional guidelines for bringing up the transmitter in the forum.

73 Steve K1EL

Archive for March, 2011

openQRP Site now reopened for registration

March 7th, 2011

After fixing some problems with the website, I have re-enabled openQRP registration. The only way I have been able to effectively screen members is with an invitation code. If you would like to join the group please [email me](#) and I will send you the code which you will then enter as part of the registration process.

73

Steve K1EL

Archive for April, 2011

Day of Reckoning

April 27th, 2011

I am at the point where the cost of the kit has been determined. It was based on actual parts cost with an allowance to cover the labor I will need to pay someone to do the actual production kitting.

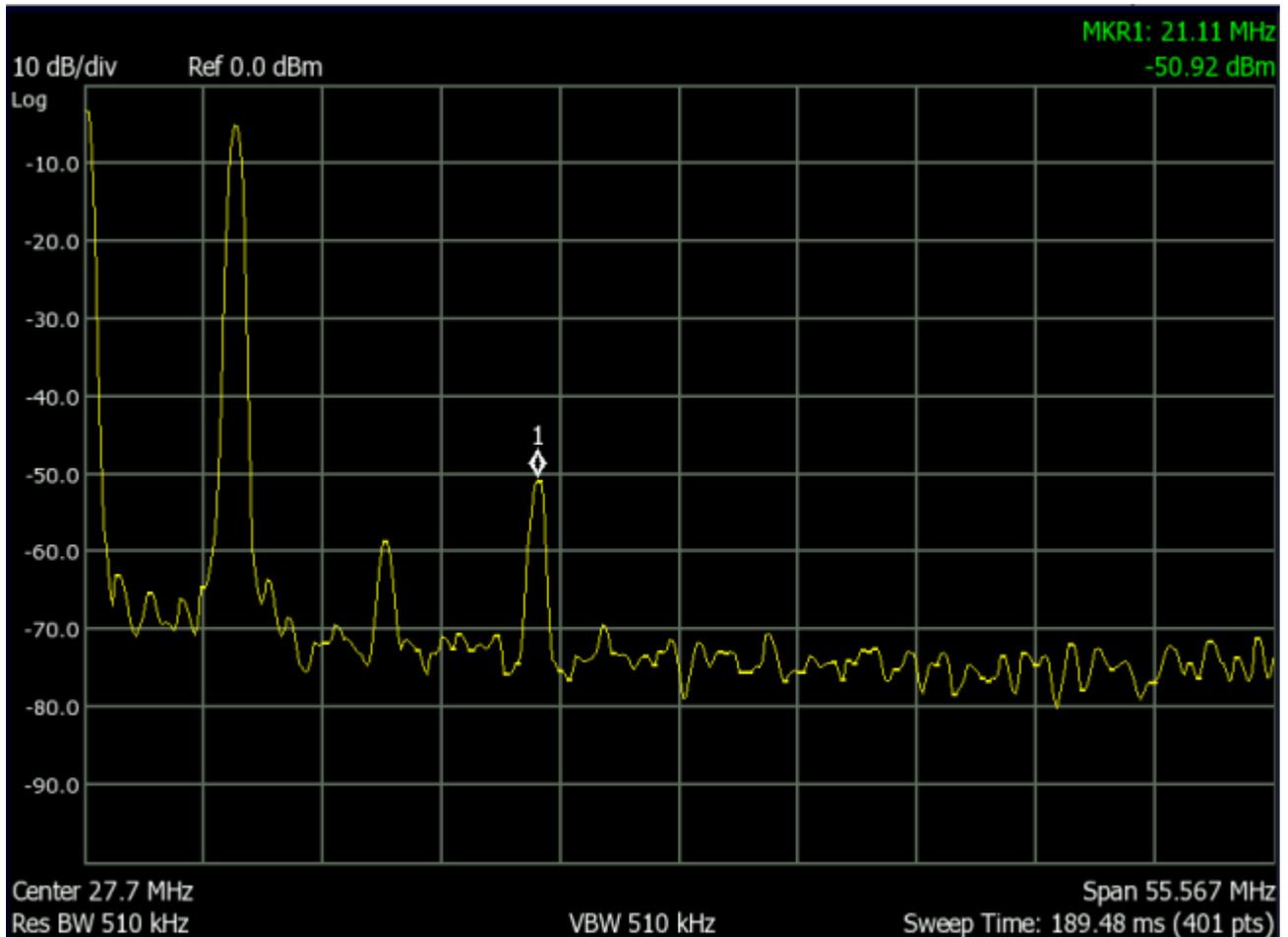
The cost for a kit, including all components, backlit display, and enclosure, is \$149. USPS Priority Mail shipping cost is an additional \$11. International shipping charge will be an additional amount that I have not determined yet, whatever the post office charges that will be the cost.

So I call this the day of reckoning because I have to see if the interest justifies continuing. I have 18 kits that I will be distributing over the next month. I hope to send these primarily to those who are interested in working on the firmware. I will be upgrading the firmware functionality myself for these 18 kits, but I was hoping to get some help to enhance it even further.

If interest justifies it, we will have another 75 kits available in about two months which will have the benefit of more improved firmware.

If you have a genuine interest in buying a kit please [email me](#) and let me know so I can formulate a plan. In addition if you are genuinely interested in being a firmware developer on the radio please let me know as soon as you can so I can begin to allocate the 18 kits in queue.

For general interest, here is a shot of the RF spectrum that I took today. Comparing back to February 2010, you can see the improvements that the transmitter clean up and bias control provide.



73 Steve K1EL

Assembly Guide Preview

April 20th, 2011

It's not finished yet but I decided to post the assembly guide so folks could have a preview. Look on the right sidebar for the link to the assembly guide.

Here's the direct link: http://www.openqrp.org/files/openQRP_ASSY_R03.pdf

Parts List Updated

April 19th, 2011

I'm going piece by piece, today I uploaded the new parts list to go along with the R03 schematics. For reference, the new PCB revision is REV D. The doc files that go with this PCB are version R03. I guess it's a little confusing.

The next post will be the assembly guide which will probably take a day or two more. It's a big job.

Folks have asked if there is anything to do, one easy thing would be to compare the schematics with the parts layout and layout guide grid. Now is the time to clear up any discrepancies. Also, since I have posted the latest firmware directory, it would be an excellent exercise to load that up and do a test build with the latest Arduino tools. I am still on Arduino version 21, I am not sure where they are now but I'm sure I'm at least a couple of versions behind. Look back to October 7th for a detailed description of getting the firmware tools up and running.

I'm trying to get comments working again on this blog, it was working a while ago but I broke something along the way. It's much easier to post a quick comment on the blog rather than go to the forum so I want to get that going again.

73 Steve K1EL

Design Files Update

April 18th, 2011

Today I updated the following three files on the Design Files/Tools Page:

Firmware: CustomHardware directory: CustomHardware_04_18_11.zip

Schematics: openQRP_SCH_R03.zip *Please use latest version of TinyCAD (2.80.03 or newer)*

PCB Layout: openQrp_PCB_R03.zip

These files reflect the latest versions. The side bar link for the pdf schematics has also been updated.

Next on the list for updating will be:

- 1) Parts List with locator guide
- 2) New Version of the Assembly Guide

73 Steve K1EL

Project Update – April 2011

April 16th, 2011

Due to other commitments I have not been able to devote much time to the 'OQ' project for a couple of months. I have several emails asking about the radio and the current status.... Here it is:

- 1) Four beta kits were sent out and during the build a couple of issues were found. To make a long story short I decided to make a few changes to the PCB layout to add an adjustable Tx final bias and add backlight control for the LCD display. We had changed the [K42 kit display](#) to one with backlight and I bought extras for the QRP radio project. So now with stable Tx power and a backlit display the two nagging problems uncovered in the beta build have been addressed. I have built one of the new version radios and in the process created a very detailed assembly guide incorporating all

of the feedback received from the beta builders. I am just putting the finishing touches on that now and will post it along with the updated schematics and parts list.

2) Firmware: I have not made much progress on updating the firmware. As a matter of fact the firmware drop on the tools page is sadly out of date. I will at least get the latest build up there this week. I am hoping that I can get some Arduino whizzes to contribute to the firmware but I know it's hard to do without a radio. So my next batch of kits will be aimed at those who have an interest in firmware development. I am preparing a run of 30 kits to go out mid-May and that is my focus at the moment.

I'll post some pictures of the new version showing how nice the backlit display is later this week.

73 Steve K1EL

Archive for May, 2011

Firmware Work

May 31st, 2011

I have been finalizing the radio firmware for initial release. I had hoped to get some help but ended up doing it myself and it has taken a while.

I added a setup function that makes Rx and Tx alignment very easy, no extra test equipment is required. Also included a frequency display calibration utility. It used to take about 45 minutes to calibrate the frequency display, set Tx offset, and dial in the Rx BFO, now it takes about 10 minutes, no o-scope required.

New menu driven options makes operation of the radio, in general, much easier. For example the user can now scroll through a list of options and select using the front panel push buttons. Tune, RIT, and keyer speed can still be set directly and quickly by pressing two button combinations. All options are stored in EEPROM so that they are reloaded every time the radio is turned on.

The keyer now works fine (there was an interrupt timing bug) but I am still fine tuning the Rx/Tx/muting delays, I'll have to sit down with a scope and dial them in for minimal thump.

The assembly manual is looking very good, I did one final step reordering. The assembly just didn't flow smoothly and there were a couple of steps that were very difficult to do after most of the parts were on the board.

I still continue to have fun making contacts and collecting good signal reports.

73

Steve K1EL

Radio Fun

May 8th, 2011

Well, I took the weekend off and instead of working on the radio I decided to make some contacts with it. I lurked around the New England QSO Party and worked some stations there. Tonight I just did some general calling and answering CQs and did quite well. Actually just finished up a QSO with

an SM6 ! Very nice opening to Sweden on 40 meters. Just using a vertical with 6 watts of OQ power. I like the CW reader, it's certainly a novelty and after an hour or so I went back to head copy, but it's still fun to challenge it to see what it will or won't copy.

I am not at all happy with the keyer though, it kept locking up on me, so I am going to rip it out and put my K1EL keyer back in for now. I will revisit the situation later, hopefully it's some minor implementation bug on my part. There's a few other operational things that need tweaking as well, the kind of stuff you discover while making contacts. All and all I am very happy and can't wait to get more of these rigs out there.

GN de K1EL

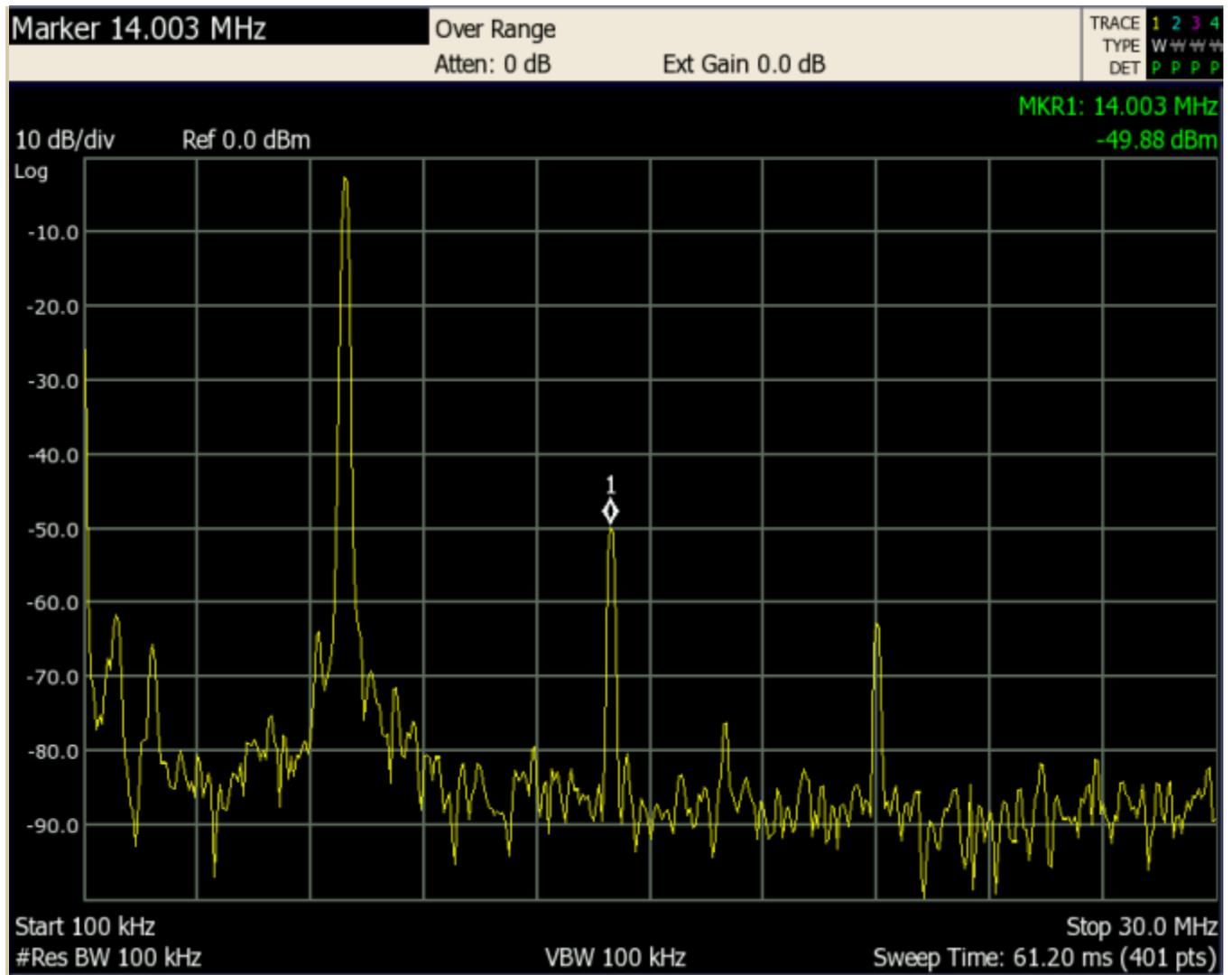
Rerun of Spectrum Plot

May 6th, 2011

Something bothered me about the plot I posted the other day. The fact that the third harmonic was almost twice the amplitude of the second harmonic didn't make a lot of sense to me but I didn't have time to think about it then.

While looking over the radio this weekend and taking pictures for the assembly manual, I noticed that the gate termination resistor on the IRF510 final was not populated. I took a look at the output wave form, pre-low pass filter, and it was ringing like crazy, right around 21 MHz. The low pass filter knocked this down considerably but it was not the way it should be.

So I put in the 12 ohm gate termination resistor and the output wave form cleaned up nicely. This was because the impedance match between the driver and gate circuit was much better. I got some time on the Spectrum Analyzer at work today and reran the sweep and now it makes sense, as you will agree:



On the Subject of Spectral Purity Requirements, FCC Regulations State:

97.73 ...the mean power of any spurious emission or radiation from an amateur transmitter, transceiver, or external radio frequency power amplifier being operated with a carrier frequency below 30 MHz shall be at least 40 dB below the mean power of the fundamental without exceeding the power of 50 mW. For equipment of mean power less than five watts, the attenuation shall be at least 30 dB.

From the plot above:

Fundamental level is at -2.6 dbm, 2nd harmonic is at -49.88 dbm, and 3rd harmonic is at -62 dbm.

Closest spur is 47.28 dbm below fundamental, lots of margin there.

<Edit 5-9-2011>

I did a bit more experimenting with the final's gate termination with the idea of moving the cut off frequency well above 7 MHz. A 10 ohm in series with .1 uF really knocks the 7 MHz drive down too much. I settled on a 4.7 ohm resistor in series with .001 uF. This performs very well bringing the power output up a watt or two while still meeting spectral purity requirements. (Plot above was taken with new network in place) Driving the gate a little harder results in the IRF510 being in saturation longer which reduces the MOSFET's power dissipation. I can actually run without a heatsink now. I won't eliminate it but I may make it smaller and easier to assemble.

Schematics and Part List Update

May 4th, 2011

Many thanks to Don VE1AOE for his careful review of the OQ schematics and parts list. He found quite a few errors which he reported and I fixed. I posted the updated files today.

I have been working on updating the OQ firmware to add a set of radio test/calibration utilities and a menu driven settings capability. The last thing slated for addition will be a better CW message editor. This will be released as Revision A firmware. I am going to skip the firmware beta kits and ship radios to anyone who would like one. Release A firmware will be complete and provide all the basic functions required to test and run the radio. More bells and whistles can be added later.

73 Steve K1EL

Archive for June, 2011

Documentation Updates

June 14th, 2011

I have posted new revisions of the Assembly Guide and Parts List. Those who are familiar with previous versions will see a substantial improvement. Included in this revision are details on the new firmware additions. I have made changes to the assembly order to improve the flow. There were a couple of rough spots before that I needed to change.

So for all practical purposes the documentation package is complete and I can start sending out some kits. I will start with kit requests I have received by email and then open it up to everyone. I will be sending out upgrade kits to the beta builders to allow them to bring their radios up to the current version.

As a side note I completed a 30 meter version of the radio which works great. The spectrum plot is as good as the 40 meter version. I don't get quite as much power out, about 5 watts instead of 6 watts but I still have some adjustments to make. I don't have any immediate plans to release a 30 meter kit, I mostly wanted to see how well the design scales to other bands. I also have plans to build a 20 meter version.

73
Steve
K1EL

Archive for September, 2011

Final Documentation Update

September 14th, 2011

I posted the final updates to the transceiver documentation package. The Schematics, Parts List, and Assembly Guide have all been finalized and posted. As always, even though I have reviewed the docs many times, there is a possibility that there may still be a typo.

Steve K1EL