pe1nnz.nl.eu.org

some thoughts on digital signal processing in amateur radio ...

Google Search

● pe1nnz.nl.eu.org

Saturday, January 26, 2019

QCX-SSB: SSB with your QCX transceiver

Over Christmas I have been playing around with a simple modification that transforms the QCX into a Class-E driven SSB transceiver. With this setup I have been able to make several SSB contacts and FT8 exchanges across Europe and so far this experiment is working reasonable well. It can be fully-continuous tuned through bands 160m-10m in the LSB/USB-modes with a 2200Hz bandwidth, provides up to 5W PEP SSB output and has a software-based full Break-In VOX for fast RX/TX switching in voice and digital operations.

The SSB transmit-stage is implemented in a completely digital and software-based manner: at the heart the ATMEGA328 is sampling the input-audio and reconstructing a SSB-signal by controlling the SI5351 PLL phase (through tiny frequency changes over 800kbit/s I2C) and controlling the PA Power (through PWM on the key-shaping circuit). In this way a highly power-efficient class-E driven SSB-signal can be realized; a PWM driven class-E design keeps the SSB transceiver simple, tiny, cool, power-efficient and low-cost (ie. no need for power-inefficient and complex linear amplifier with bulky heat-sink as often is seen in SSB transceivers).

An Open Source Arduino sketch is used as the basis for the firmware, a hardware modification bypasses the QCX CW filter and provides a microphone input in-place of the DVM-circuit; the mod is easy to apply and consist of four wires and four component changes and after applying the transceiver remains compatible with the original QCX (CW) firmware.

This experiment is created to try out what can be done with minimal hardware; a simple ATMEGA processor, a QCX and a bit of signal processing to make SSB in an artificial manner. It would be nice to add more features to the sketch, and see if the QCX design can be further simplified e.g. by implementing parts of the receiver stage in software. Feel free to experiment with this modification and let me know your thoughts or contribute here: https://github.com/threeme3/QCX-SSB

More discussion here: https://groups.io/g/QRPLabs/topic/29572792

Blog Archive

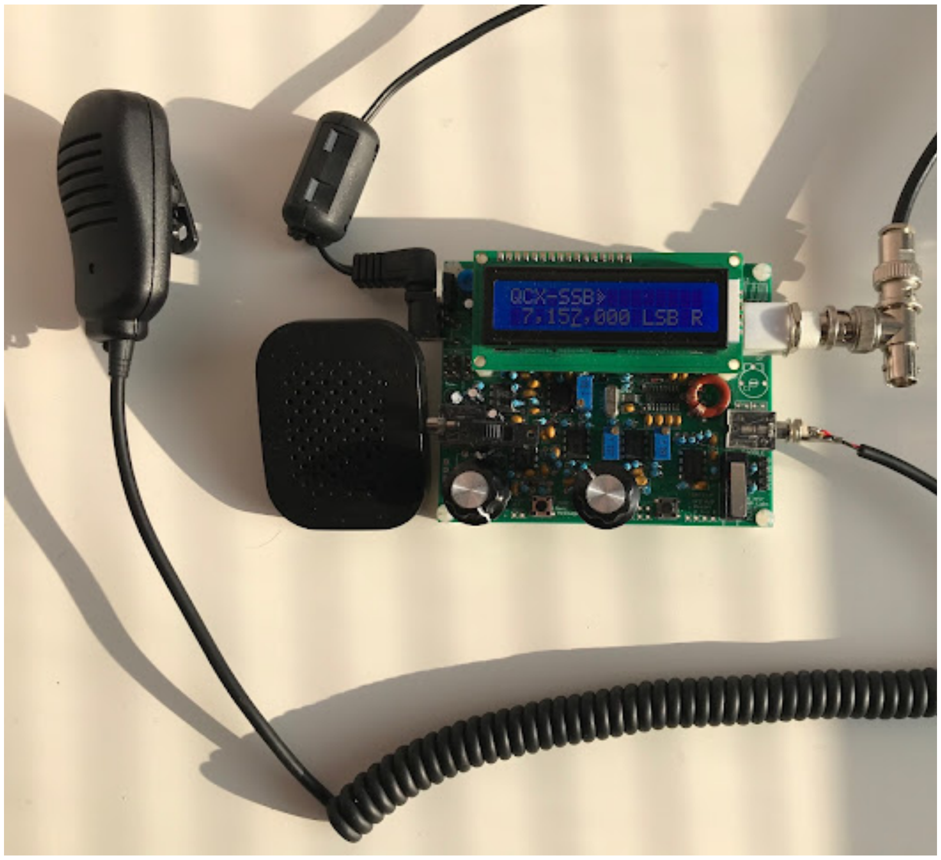
- ▼ 2019 (1)
  - ▼ January (1)
    - QCX-SSB: SSB with your QCX transceiver
- ► 2013 (1)
- ► 2012 (3)
- ► 2009 (6)

About Me

I'm Guido, radio amateur licensed as PE1NNZ, living in south of Netherlands, i like qrp, homebrewing, digital communications, being reachable by email at pe1nnz@amsat.org

Links

- aa1tj
- aa1tj-fb
- aa7ee
- ag1le
- aj4vd
- ak2b
- aprs.fi
- bellard
- blogger
- cave
- ce
- cnlohr
- cqham.ru
- cw
- daru
- dj7oo
- dk7ih
- dk7jd
- dl1adr
- dm3mat
- dmitry
- dxheat



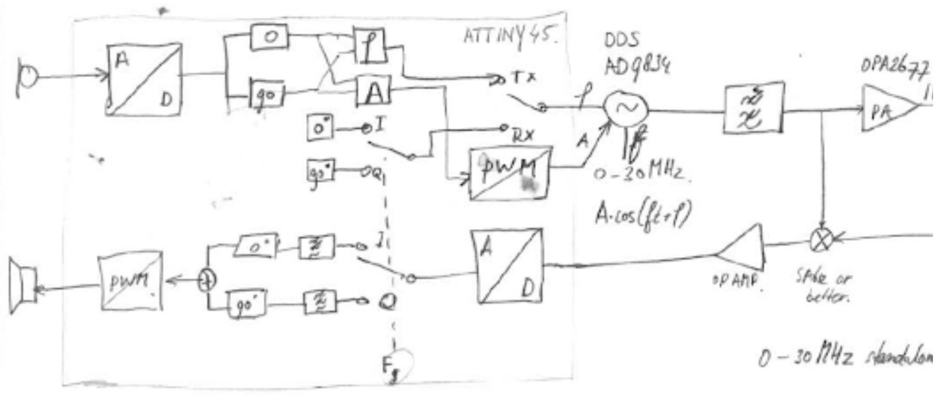
Posted by threeme3 at 1:07 PM No comments:

Wednesday, May 8, 2013

Direct SSB generation by frequency modulating a PLL

The following code can generate SSB modulation just by controlling a PLL carrier. I have applied this method on the RapsberryPi PLL, and made several contacts on 40m and 20m band with my RaspberryPi.

I will explain the principle in more detail, later. The original idea is depicted here:



The original idea (as seen above) was to implement a complete SSB modulator and demodulator with software in a ATTINY45 micro-controller chip and a AD9834 as DDS. The software samples microphone signal with ADC and then derives the amplitude and phase of

- dzl
- ea4gpz
- eham
- es1uvb
- f4goh
- f5nlg
- g0orx
- g3xbm
- g3zjo
- g4jnt
- g7ckf
- grabber compendium
- grabber plus
- ha8tks
- habhub
- hackaday
- hackernews
- hackgreensdr
- hackster
- hellocq
- hellocq2
- hw.cz
- ik1zyw
- ja2nkd
- jf3hzb
- justine
- k6hx
- ka7exm
- ka7oei
- ka9q
- kc3xm
- kd1jv
- kd1jv-fb
- kenshirriff
- kf6vo
- kiwisdr
- lb7ug
- linkfanel
- lu7aa
- lu7did
- m0ifa
- m0nka
- m0xer
- m0xpd
- m1kta
- m6kwh
- mikrocontroller.net
- mitxela
- n9vv
- nt7s
- oh2btg
- oh2eat
- oh2eiq
- oh2ftg
- ok1aik
- on7dq
- openwebrx
- pa1dsp
- pa1gsj
- pa3fym

the audio signal. The amplitude and phase extraction is derived from the in-phase microphone signal and its 90 degree phase shifted quadrature signal (created by a Hilbert transform on the in-phase signal). The resulting phase is replicated on the oscillator signal of the DDS, and the amplitude information is used to change the drive-strength (current) of the DDS. The output of the DDS is representing a clean single side-band output SSB near the configured frequency.

To proof this idea is working, I implemented the algorithm on a RaspberryPi. In the RaspberryPi setup there is no DDS, but there is a digital-PLL on the micro-processor SoC which we can use for RF generation. By manipulating the Integer and Fractional dividers, a frequency can be generated from 0 to 250 MHz on GPIO4 pin with a resolution of about 20 Hz at 7 MHz. By alternating the Fractional divider between two nearby values, the resolution can be further increased to micro-Hz level; for this a 32 bit clock at 32 MHz is used to generate a PWM alike switching pattern between the two Fractional divider values.

The PLL oscillator can be phase modulated by short manipulations of the configured frequency. Increasing the frequency temporarily and then restoring to its original frequency, will shift the phase upwards, while decreasing the frequency temporarily will decrease the phase of the signal.
In this way the phase information for generating a SSB signal can be applied to the RaspberryPi PLL by means of frequency modulation. To do so, first phase differences are calculated for every sample. For each phase deviation (difference) the corresponding frequency change is calculated. This is dependent on the rate that the PLL is changed from frequency; the exact formula is: dev_freq_hz = dev_phase_rad * samp_rate_hz / (2 * pi)

The amplitude signal can be applied by changing the drive-strength of the GPIO4 port, which has 8 levels with a dynamic range of about 13 dB. After some experimenting, amplitude information can be completely rejected, instead the frequency must be placed outside the single side band (e.g. placed to center freq). In this case single side band is generated wihout suppressed carrier, i.e. a constant amplitude envelope is applied (good to prevent RF interference). In this case the audio quality does not seem to suffer from this constant amplitude, and the audio quality becomes even better by applying a little-bit of noise to the microphone input. My impression is that using SSB with constant amplitude increases the readability when the signal is just above the noise.

The RaspberryPi receives the Microphone input via an external USB sound device. To improve the SSB quality, the signal is companded by a A-law compression technique. Three parallel BS170 MOSFETs where directly driven by RaspberryPi GPIO4 output to create about 1Watt of RF. On 40m I could made several SSB contacts through Europe using this setup, receiving stations back by using a nearby online WebSDR receiver. From my location in south-Netherlands following contacts where made:

Apr 27 14:00 40m on4azw/p
Apr 27 14:00 40m pd6king
Apr 27 14:00 40m pa150ba
Apr 27 14:00 40m hb9ag
Apr 27 14:00 40m pd2edr
Apr 28 13:22 40m m0zag
Apr 28 13:22 40m g100rsgb
Apr 28 13:22 40m hb9efx
Apr 28 13:22 40m pa3zax
Apr 28 13:22 40m tm02ref
Apr 28 16:00 20m ea2dt
Apr 29 19:00 40m g3mlo
Apr 30 10:23 7100 pc1king
Apr 30 16:42 7082 pa2cvd
Apr 30 17:00 7092 dl9fcs Gazi, Frankfurt 57

pa3pm
pe1chl
pi1nos
pi1utr
pi4dec
pi4raz
pocorgtfo
pskreporter.de
pskreporter.info
py2ohh
r4f
reversebeacon
rm9wy
rossum
rxtx
sdr.hu
sp-hm.pl
Sparky's blog
spritemods
sv3ora
tb
technoblogy
tejeez
tejeez2
telegraphy
tj
ttrftech
ub4hao
urqrp
ve3mkc
virtuallyfun
vk2ihl
vk2zay
vk3hn
vk3ye
vk5dgr
vocap
vrza
w3jdr
w4kia
websdr.org
websdr.tk
wspr.aprsinfo
wsprd
wsprnet
xv4y
yd1osc
yl2gl
yukilab
za
zl1bpu
zl2ctm

Apr 30 18:30 7074 pa3a Arie, Barendrecht
May 4 11:52 7107 g100c
May 9 15:20 7077 dl/pc1mk Menno, Borkum

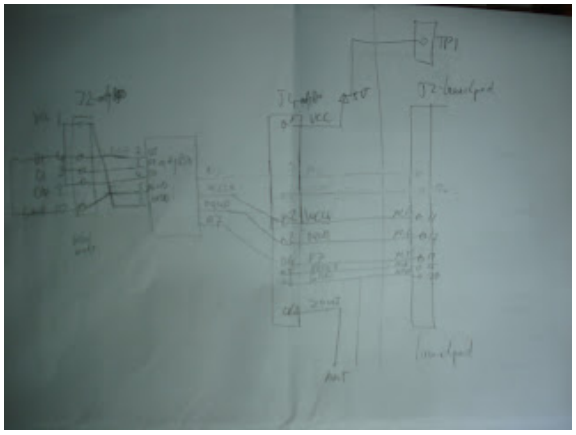
Below a fragment of the code that generates SSB signal in the form of amplitude and frequency deviation information, based on the inputed audio signal:

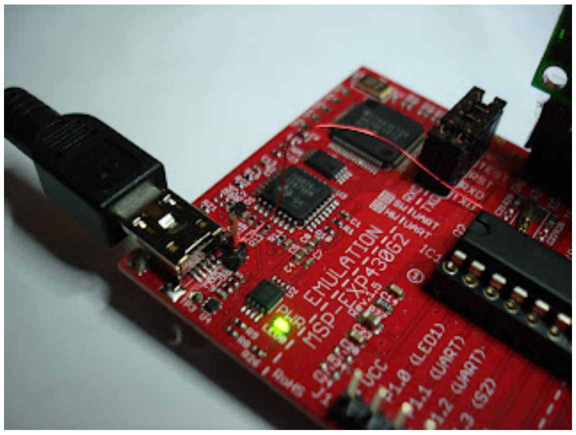
```
#define ln(x) (log(x)/log(2.718281828459045235f))

static int xv[45];

void filter(int val, int* i, int* q)
{
    int j;

    for (j = 0; j < 44; j++) {
        xv[j] = xv[j+1];
    }
    xv[44] = val;

    *i = xv[22];

    int _q = (xv[1] + xv[3] + xv[5] + xv[7]+xv[7] + 4*xv[9] +
6*xv[11] \
        + 9*xv[13] + 14*xv[15] + 23*xv[17] + 41*xv[19] + 127*xv[21] \
        - (127*xv[23] + 41*xv[25] + 23*xv[27] + 14*xv[29] + 9*xv[31]
\
        + 6*xv[33] + 4*xv[35] + xv[37]+xv[37] + xv[39] + xv[41] +
xv[43]) ) / 202;

  *q = _q;
}

int arctan2(int y, int x)
{
   int abs_y = abs(y);
   int angle;
   if(x >= 0){
      angle = 45 - 45 * (x - abs_y) / ((x + abs_y)==0?1:(x + abs_y));
   } else {
      angle = 135 - 45 * (x + abs_y) / ((abs_y - x)==0?1:(abs_y -
x));
   }
   return (y < 0) ? -angle : angle; // negate if in quad III or IV
}

static float t = 0;
static float prev_f = 0;
static float prev_phase = 0;
static float acc = 0;

void ssb(float in, float fsamp, float* amp, float* df)
{
   int i, q;
   float phase;

   t++;
   filter(in * 128, &i, &q);
   *amp = sqrt( i*i + q*q) / 128.0f;
   if(*amp > 1.0){
     printf("amp overflow %f\n", *amp);
     *amp = 1.0;
   }
   phase = M_PI + ((float)arctan2(q,i)) * M_PI/180.0f;
   float dp = phase - prev_phase;
   if(dp < 0) dp = dp + 2*M_PI;
   prev_phase = phase;

   *df = dp*fsamp/(2.0f*M_PI);
}

void main(){
    int samplerate = 8000; // set sample-rate of microphone input
here, use 4800 to 8000
    int notvi = 0;   // set no TVI to true if you want constant
carrier


    for(;;) {
        float df, amp;

        int data = //16 bit microphone input here
        ssb((float)data/32767.0, samplerate, &amp, &df);

        float A = 87.7f; // compression parameter
        amp = (fabs(amp) < 1.0f/A) ? A*fabs(amp)/(1.0f+ln(A)) :
(1.0f+ln(A*fabs(amp)))/(1.0f+ln(A)); //compand

        int ampval = (int)(round(amp * 8.0f)) - 1;
        int enaval = (ampval < 0) ? 0 : 1;

        if(notvi && ampval < 0){
          enaval = 1; // tx always on
          df = 0;
        }
        if(notvi) ampval = 7; // optional: no amplitude changes

        if(ampval>7) ampval=7;
        if(ampval<0 ampval="0;<!--0--">

       // here, send ampval to 3 bit drive-strength register (if
available)
       // here, add or substract df from PLL center frequency (in
Hz), adding will make USB, substract. LSB
   }
}
```

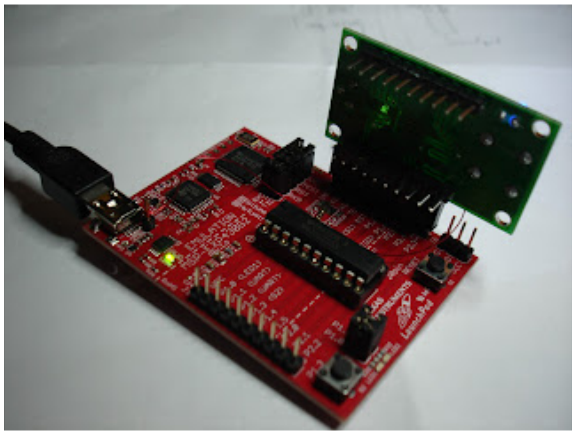
Posted by threeme3 at 6:35 AM 21 comments:

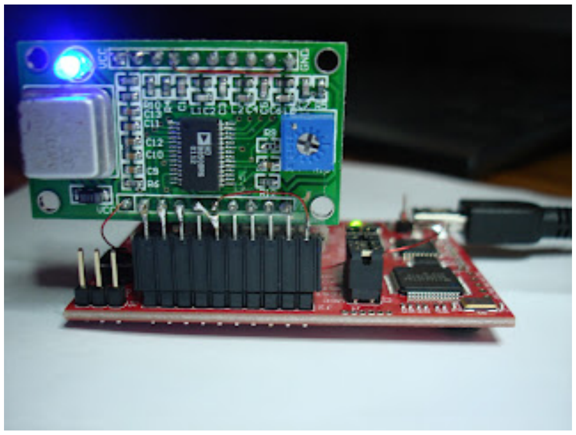
Wednesday, December 5, 2012

Launchpad AD9850 preliminary beacon experiments









Connections:

J2 of ad9850:
Connect pin 1(VCC) to pin2 (D0) and 3 (D1)
Connect pin 4 (D2) to pin 10 (GND)

J4 of ad9850:
Connect pin 1 (5v VCC) with wire to TP-1 of Launchpad (use jumper here)
Connect pin 2 (WCLK) to pin 11 (P2.3)
Connect pin 3 (FQUD) to pin 12 (P2.4)
Connect pin 4 (D7) to pin 13 (P2.5)
Connect pin 5 (RESET) to pin 15 (P1.7)
Connect pin 6 (GND) to pin 20 (GND)
Connect pin 10 to Antenna

Code used to program DDS, see http://nr8o.dhlpilotcentral.com/?p=83 and modify:

```
 #define W_CLK 11       // Pin - connect to AD9850 module word load
clock pin (CLK)
 #define FQ_UD 12       // Pin - connect to freq update pin (FQ)
 #define DATA  13       // Pin  - connect to serial data load pin
(DATA)
 #define RESET 15      // Pin - connect to reset pin (RST).
```

Posted by threeme3 at 8:48 PM No comments:

Monday, November 26, 2012

C snippet for generating Opera symbols

```
char* itoa(unsigned long val, int n, char* buf, int radix)
{
  char c[] = "0123456789abcdefghijklmnopqrstuvwxyz";
  int i = 0;
  while (n-i) {
    unsigned long d = (unsigned long)pow(radix, n-i-1);
    buf[i++] = c[val/d];
    val = val%d;
  }
  buf[i] = '\0';
  return buf;
}

int crc16op(char* msg, int n)
{
   int i,j,crc=0; // crc-16 (0x8005 poly, flip in and out, 2 zero
bytes in tail) of msg[0..n-1]
   for(i=0;i!=n+2;i++){
     if(i
     for(j=0;j!=8;j++)
       crc = crc&1 ? (crc>>1) ^ 0xa001 : crc>>1;
   }
   // replace, swap and store crc in msg[32..48]
   crc = crc&0xff00 ? ( crc&0x00ff ? crc : 0x001b|(crc&0xff00) ) :
0x2b00|(crc&0x00ff);
   crc = ((crc&0x00ff)<<8 amp="amp" crc="crc" xff00="xff00">>8);
   return crc;
}

int main(int argc, const char* argv[]){
   if(argc != 2){
     printf("usage: opera [callsign]\n");
     return 0;
   }
   const char* call = argv[1];
   char msg[239];
   int i,j;

   char c[]="      "; //align last prefix digit at c[2] and fill gaps
with blanks
   int aligned = isdigit(call[2]);
   strncpy(aligned ? c : &c[1], call, aligned ? 6 : 5);
   strupr(c);

   unsigned long n1=(c[0]>='0'&&c[0]<='9'?c[0]-'0'+27:c[0]==' '?
0:c[0]-'A'+1); // packing call e.g. " K1JT ", "AA1AA " into n1
   n1=36*n1+(c[1]>='0'&&c[1]<='9'?c[1]-'0'+26:c[1]-'A');
   n1=10*n1+c[2]-'0';
   n1=27*n1+(c[3]==' '?0:c[3]-'A'+1);
   n1=27*n1+(c[4]==' '?0:c[4]-'A'+1);
   n1=27*n1+(c[5]==' '?0:c[5]-'A'+1);
   itoa(n1, 28, &msg[4], 2); //msg[4..32] = binary representation n1
in ASCII
   itoa(crc16op(&msg[4], 28), 16, &msg[32], 2); //store bin-crc of
msg[4..31] in msg[32..47]
   itoa(crc16op(&msg[4], 28+16) & 0x07, 3, &msg[48], 2); //store bin-
crc of msg[4..47] in msg[48..51]
   msg[0]=msg[1]=msg[2]=msg[3]='0';  //unused bits msg[0-3]

   const char* prn_vec =
"1110000101010101111110011011010000000110010001010101 1";
```

```
   for(i=0; i!=51; i++) //scramble
     msg[i] = ((msg[i]-'0') ^ (prn_vec[i]-'0')) +'0';

   const char* wh[] = {"0000000", "1010101", "0110011", "1100110",
"0001111", "1011010", "0111100", "1101001"};
   for(i=(51/3)-1; i>=0; i--){  // Walsh-Hadamard encoding to
msg[0..118]
     char b = (msg[i*3+0]-'0')*4+(msg[i*3+1]-'0')*2+(msg[i*3+2]-
'0')*1;
     for(j=0; j!=7;j++)
       msg[i*7+j] = wh[b][j];
   }

   for(i=0; i!=7; i++)  // interleave 7x17 to msg[121..240]
     for(j=0; j!=17; j++)
       msg[121+j+17*i] = msg[i+7*j];

   for(i=0; i!=119; i++){  // Manchester encoding to msg[2..120]
     msg[2*i+1+2] = msg[121+i];
     msg[2*i+0+2] = (msg[2*i+1+2] == '0') + '0';
   }
   msg[0] = msg[1] ='1'; // head
   msg[239] = '\0'; // tail

   printf("message=%s symbols=%s\n", c, msg);
}
```

Posted by threeme3 at 9:57 PM No comments:

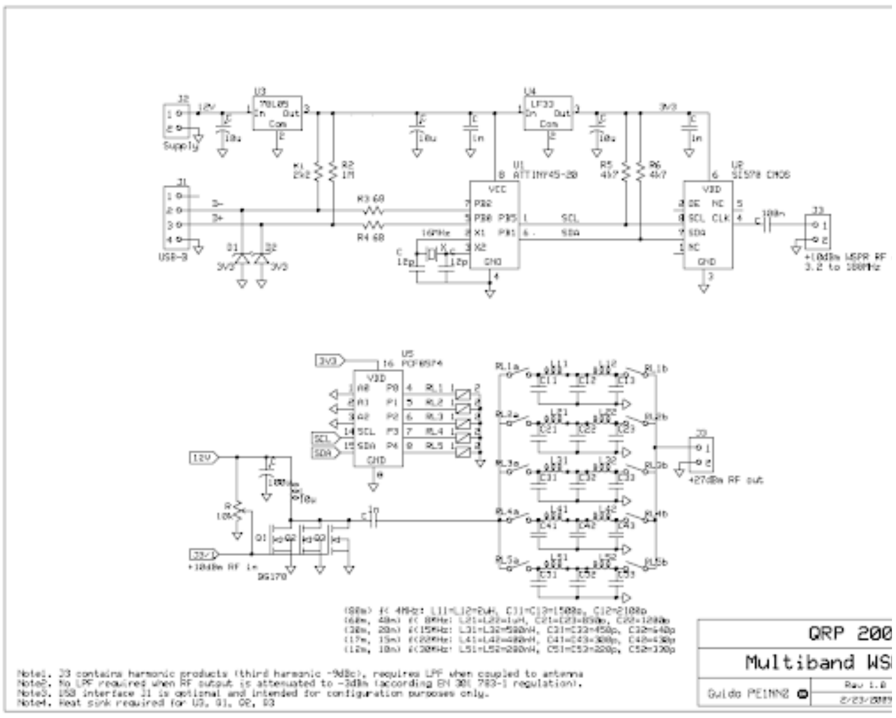
jt65-source code

As it seems the author has lost the source code of the original jt65-hf version 1093, here is a link to the source code: http://sdrv.ms/PbOz48

Posted by threeme3 at 9:55 PM No comments:

Thursday, March 12, 2009

Multiband SI570 WSPR beacon



QRP 200
Multiband WS
Guido PE1NNZ
Rev 1.0
2/23/2009
(80m) f< 4MHz: L11=L12=2uH, C11=C13=1500p, C12=2100p
(60m, 40m) f< 8MHz: L21=L22=1uH, C21=C23=850p, C22=1200p
(30m, 20m) f<15MHz: L31=L32=500nH, C31=C33=450p, C32=640p
(17m, 15m) f<22MHz: L41=L42=400nH, C41=C43=300p, C42=430p
(12m, 10m) f<30MHz: L51=L52=280nH, C51=C53=220p, C52=330p
Note1. J3 contains harmonic products (third harmonic -9dBc), requires LPF when coupled to antenna
Note2. No LPF required when RF output is attenuated to -3dBm (according EN 301 783-1 regulation).
Note3. USB interface J1 is optional and intended for configuration purposes only.
Note4. Heat sink required for U3, Q1, Q2, Q3
+10dBm WSPR RF 3.2 to 100MHz
+27dBm RF out
+10dBm RF in
ATTINY45-20
SI570 CMOS
PCF8574
78L05
LF33
BS170

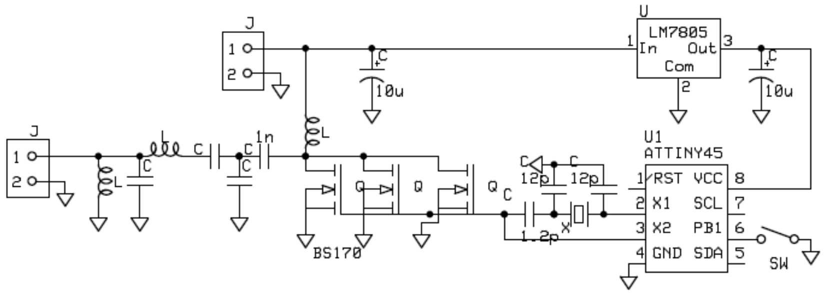
C:\temp\x\wsprbcn570.sch - Sheet1

more info here (login account required): http://wsprnet.org/drupal/node/2043

Posted by threeme3 at 2:14 PM No comments:

Wednesday, March 11, 2009

Idea for a simplistic WSPR beacon



update: similar idea here: http://rheslip.blogspot.nl/2011/12/tiny-beacon.html

Posted by threeme3 at 4:32 PM No comments:

Home Older Posts

Subscribe to: Posts (Atom)