A Very Simple PTS160-based JT Mode Beacon
By Roger Rehr, W3SZ 4-3-2024

1. This paper describes using a PTS (Programmed Test Sources) synthesizer either directly or as a signal to be either multiplied or mixed or both to provide a higher frequency JT mode beacon. In this example we will create a Q65-60C beacon using a PTS-160, but the method applies generally to the JT modes and submodes and can be used with PTS synthesizers other than the PTS-160 with just slight modifications of the code used to control the PTS synthesizer. In those cases where frequency multiplication of the PTS signal is used the code will adjust the frequency spacing of the JT-mode tones so that tones will have the appropriate frequency spacing after frequency multiplication. In any case, the frequency resolution of the PTS synthesizer used will need to be sufficient to accommodate the tone spacing of the JT mode being used.

2. Our method starts with the creation of an "itone" file for the desired mode and message. An itone file is a time-ordered list of the message tones sent during one T/R cycle, with the tone values ranging from 0 to x where x is one less than the number of different tones that the mode uses. For Q65, this number is of course 64. The length of this list, termed the symbol length, is the number of tone intervals that are used by the mode in sending a message, and for Q65 this is 85.

3. The WSJT-X distribution's set of files contains a command-line executable file named q65code.exe. This file uses the command structure

```
q65code.exe "message" > itones_Q65.txt
```

to generate an itone file named itones_Q65.txt containing the itones representing the given message. An example of such a file is here, for the message "W3SZ/B FN20AG":

```
-----file begins-----
 Generated message plus CRC (90 bits)
6 bit :   27   2  41  42   6  43  61  45  27  24  19  54   0   4  10
binary:
011011000010101001101010000110101011111101101101011011011000010011110110000000000100
0001010

 Codeword with CRC symbols (65 symbols)
 27  2 41 42  6 43 61 45 27 24 19 54  0  4 10  4 44  6  0 43
 43 51 28  4 48 32 13 39 23 59 49 49 62 22  5 13 38 17 20 20
 32 60 55 53  3 42 38 44 51 14 15 41 41  5 27  5 30  7 53 54
 19  8 34 32 18

 Channel symbols (85 total)
  0 28  3 42 43  7 44 62  0 46 28  0  0 25  0 20 55  1  5 45
  7  0  0  1 44  0  0 44 52 29  5 49  0 33  0 14 40  0 24 60
 50 50 63 23  6  0 14 39 18  0 21 21 33 61  0 56 54  4 43  0
 39  0 45 52 15  0 16 42  0 42  6 28  6  0 31  0  8 54 55 20
  9 35 33 19  0
-----file ends-----
```

All we need from this file for our purpose is the 85 channel symbols, in csv format. Each element of this set of channel symbols represents the frequency offset of a given itone from the base frequency, with that offset specified by the individual list value times the tone spacing. In the case where there is to be no frequency multiplication of the PTS-160 signal, and if the tone spacing is 0.75 Hz, the first tone in the above list would be offset by 0 Hz, the next tone offset by 0.75 * 28 = 21 Hz, etc. If the PTS frequency were to be multiplied by 9, then the tone spacing for the signal generated by the PTS in this case would need to be 0.75 / 9 = 0.0833333 Hz so that after frequency multiplication the spacing would be the required 0.75 Hz.

As was noted in the first paragraph above, the synthesizer resolution needs to be adequate for the tone spacing of the mode being used. Tone spacings of the various JT modes are given in tables 7 thru 9 of the WSJT-X 2.7.0-rc4 user guide at https://wsjt.sourceforge.io/wsjtx-doc/wsjtx-main-2.7.0-rc4.html#SLOW_MODES. The tone spacings for the various Q65 submodes are shown the table below:

T/R Period (s)	A Spacing (Hz)	B Spacing (Hz)	C Spacing (Hz)	D Spacing (Hz)	E Spacing (Hz)
15	6.67	13.33	26.67	N/A	N/A
30	3.33	6.67	13.33	26.67	N/A
60	1.67	3.33	6.67	13.33	26.67
120	0.75	1.5	3	6	12
300	0.29	0.58	1.16	2.31	4.63

PTS Synthesizers can be obtained with resolutions from 0.1 Hz to 100 kHz. Although the PTS product code, which is affixed to the rear panel of every PTS synthesizer, indicates the resolution of the PTS synthesizer at the time of sale, many of the synthesizers available today have been modified and so the actual resolution of any given synthesizer may be either worse than or better than that indicated by the product code. For example, for a PTS synthesizer with product code 160M7O1C, the "7" indicates that the synthesizer has 0.1 Hz resolution as you can see in the table below:

RESOLUTION / PTS 310 TYPE / PTS x 10 FREQ. RANGE

Code	PTS 040/120/160/250/500/620/1600/3200/6400	PTS 310	PTS x10
0	________	________	0.1-10 MHz
1	100 KHz Resolution	Type 1	10-20 MHz
2	10 KHz Resolution	Type 2	20-30 MHz
3	1 KHz Resolution	________	30-40 MHz
4	100 Hz Resolution	________	40-50 MHz
5	10 Hz Resolution	________	50-60 MHz
6	1 Hz Resolution	________	60-70 MHz
7	0.1 Hz Resolution	________	70-80 MHz
H*	DDS with 0.1 Hz Resolution	________	
J**	DDS with 1 Hz Resolution	________	
K	DDS with 0.1 Hz Resolution	________	
8	________	________	80-90 MHz
9	________	________	90-100 MHz

*standard resolution on PTS D310, D620
standard resolution on PTS 1600, 3200, 6400; ***not available on other models*

You can see from the above that while a synthesizer with a resolution of 1 Hz should be adequate for Q65-60C which has tone spacing 6.67 Hz, that if the PTS signal frequency is being multiplied by 9, then the resulting 9 Hz resolution at the multiplied frequency would be inadequate. For this project I used a spare PTS-160 that I had on hand, with product code 160SKO20. The above table shows that the “K” signifies that this unit has a DDS synthesizer with 0.1 Hz resolution, so it would be adequate for creating a Q65-60C beacon even with relatively large frequency multiplication factors.

4. The channel list output produced by q65code.exe is not in csv form and it contains extraneous information, so this data needs to be slightly modified to the following format so that we can use it with the code that we have written:

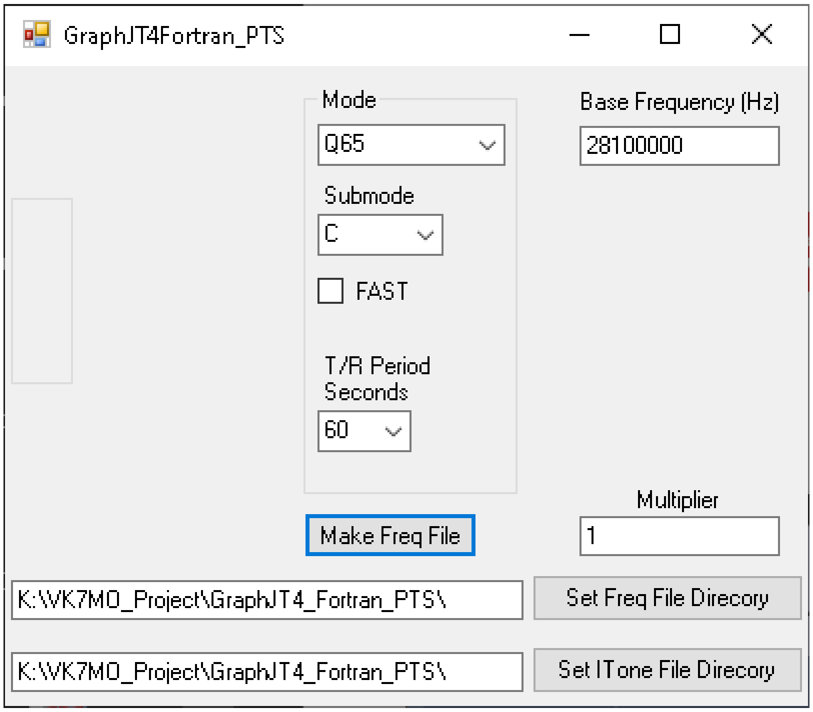
```
0,28,3,42,43,7,44,62,0,46,28,0,0,25,0,20,55,1,5,45
,7,0,0,1,44,0,0,44,52,29,5,49,0,33,0,14,40,0,24,60
,50,50,63,23,6,0,14,39,18,0,21,21,33,61,0,56,54,4,43,0
,39,0,45,52,15,0,16,42,0,42,6,28,6,0,31,0,8,54,55,20
,9,35,33,19,0
```

This formatting can be done relatively easily. using either a text editor or spreadsheet software to replace spaces with commas. CSV-format itone files which can be used directly without reformatting can also be produced by a suitably modified version of WSJTX or by a separate program.

5. Once the itones file has been suitably formatted, we can use it with my C# program GraphJT4Fortran_PTS to generate a frequency list that can then be used by an SBC such as an Arduino MEGA 2560 to control a PTS so that it sends a Q65 (or other JT-mode) message repeatedly.

6. GraphJT4 takes as user input the following parameters:
Mode
Submode
T/R Period (seconds)
Base Frequency (Hz)
Multiplier (Integer)
ITone directory (input; file name must be of the form “itones_XXX.txt” where XXX is the mode)
Freq File directory (output)

Mode, Submode, and T/R Period are selected by pull-down menus.
Base Frequency (in integer Hz) and Multiplier (an integer) are supplied by the user via text boxes.
The two directories are selected using the standard Windows FolderBrowserDialog routine.
The itone file must be named “itones_XXXXXX.txt” where “XXXXXX” is the mode name, such as "Q65", "FT8", "JT9", "JT65", etc, as the program uses the file name to select which itone file in the selected folder is used for the current calculation. The set of acceptable values for “XXXXXX” is the same as the set of modes contained in the mode pull-down list. The GUI for this program is shown below:



The Arduino sketch that I wrote expects the frequency list to be an array of 85 strings, with each string having length 10 and with each string providing the frequency value for that tone in tenths of a Hz, expressed as an integer. So a frequency of 144.290 MHz would be given as “1442900000”, for example. My program GraphJT4Fortran_PTS generates the frequency list needed by the Arduino.

Lets look at an example using a base frequency of 144033333.3 MHz. If we want to use the PTS to create a 2M beacon at this frequency, then we would run GraphJT4Fortran_PTS with selections for mode Q65-60C and base frequency 144033333.3 Hz and multiplier 1. The frequency file generated by the program will be named FreqFile_Q65-60C_85_0.600_144033333.3_1.csv and contain the following values:

"1440343333","1440345200","1440343533","1440346133","1440346200","1440343800","1440346266","1440347466","1440343333","1440346400","1440345200","1440343333","1440343333","1440345000","1440343333","1440344666","1440347000","1440343400","1440343666","1440346333","1440343800","1440343333","1440343333","1440343400","1440346266","1440343333","1440343333","1440346266","1440346800","1440345266","1440343666","1440346600","1440343333","1440345533","1440343333","1440344266","1440346000","1440343333","1440344933","1440347333","1440346666","1440346666","1440347533","1440344866","1440343733","1440343333","1440344266","1440345933","1440344533","1440343333","1440344733","1440344733","1440345533","1440347400","1440343333","1440347066","1440346933","1440343600","1440346200","1440343333","1440345933","1440343333","1440346333","1440346800","1440344333","1440343333","1440344400","1440346133","1440343333","1440346133","1440343733","1440345200","1440343733","1440343333","1440345400","1440343333","1440343866","1440346933","1440347000","1440344666","1440343933","1440345666","1440345533","1440344600","1440343333"

The frequency file name specifies, in order and separated by the underscore character '_' the following:
mode and submode: Q65-60C
symbol set length: 85
symbol duration: 0.6 (seconds)

base Frequency: 144033333.3 (Hz)
multiplier (1)

The frequency file filename provides easy access to the values of the symbol set length and symbol duration used in the Arduino sketch and makes it easy to put these values into the sketch when updating it.

If we use the same base frequency but with a multiplier of 9, this would create a beacon signal placed at 1296.300 MHz which is of course (within 1 Hz) equal to 144033333.3 * 9 Hz. The base tone of the Q65 signal will have an audio frequency of 1000 Hz with the receiver set to 1296.300 MHz and tone spacing at 1296.300 MHz will be appropriate for Q65-60C. In this case the output file will be named FreqFile_Q65-60C_85_0.600_144033333.3_9 and the contents will be:

"1440334443","1440334650","1440334465","1440334754","1440334762","1440334495","144033476
9","1440334902","1440334443","1440334784","1440334650","1440334443","1440334443","1440334
628","1440334443","1440334591","1440334850","1440334450","1440334480","1440334776","14403
34495","1440334443","1440334443","1440334450","1440334769","1440334443","1440334443","144
0334769","1440334828","1440334658","1440334480","1440334806","1440334443","1440334687","1
440334443","1440334547","1440334739","1440334443","1440334621","1440334887","1440334813"
,"1440334813","1440334910","1440334613","1440334487","1440334443","1440334547","14403347
32","1440334576","1440334443","1440334599","1440334599","1440334687","1440334895","144033
4443","1440334858","1440334843","1440334473","1440334762","1440334443","1440334732","1440
334443","1440334776","1440334828","1440334554","1440334443","1440334562","1440334754","14
40334443","1440334754","1440334487","1440334650","1440334487","1440334443","1440334673","
1440334443","1440334502","1440334843","1440334850","1440334591","1440334510","1440334702
","1440334687","1440334584","1440334443"

7. This data is inserted into an Arduino sketch written by me (named PTS_LO_LeadsParalleled.ino) that varies the PTS output frequency appropriately in order to produce each of the 85 Q65 tones required for a complete Q65-60C message.

8. Because the start of each JT-mode message must be accurately timed to the beginning of each minute, the Arduino also needs GPS input in order to start each message sequence appropriately at this time. This GPS timing is provided by a Goouuu Tech GT-U7 GPS module, which can be obtained on Amazon for less than $15.00.

9. Although the Arduino program uses the GPS signal to initiate the message at the start of each minute, the program uses the “millis” parameter produced by the Arduino for more accurate timing within the Q65 message. This avoids the need to repeatedly read and process the time message from the GT-U7 during message transmission. The millis parameter is an unsigned long integer, and represents the number of milliseconds that have elapsed since the Arduino was started. As the largest possible value for an unsigned long integer on the Arduino is 4,294,967,295, this number will “roll over” every 47.71 days. When that happens, the Arduino will send the message “reset” in frequency-shift-keyed Morse code and then return to usual operation.

10.   The PTS160 requires BCD (Binary Coded Decimal) frequency input. The Arduino sketch takes care of converting the decimal frequency input to BCD and then sends the BCD data to the 50-pin Centronics female jack on the back of the PTS-160. The frequency control inputs to the PTS-160 are somewhat complicated, as is shown in the diagram below:

TABLE 1. PTS 160 REMOTE FREQUENCY/LEVEL CONTROL

Amphenol 57-40500—On Equipment
Amphenol 57-30500—Required to Control

Digit		Weight: 1	2	4	8	Latch Enable
		Pin Numbers				
10 MHz	(0-15, Hexadecimal)	15	16	40	41	23
1 MHz	(0-9, BCD; 10-15 Invalid)	17	18	19	20	24
100 KHz		1	2	26	27	24
10 KHz		3	4	28	29	25
1 KHz		5	6	30	31	25
100 Hz		7	8	32	33	46
10 Hz		9	10	34	35	46
1 Hz		11	12	36	37	47
0.1 Hz		13	14	38	39	47

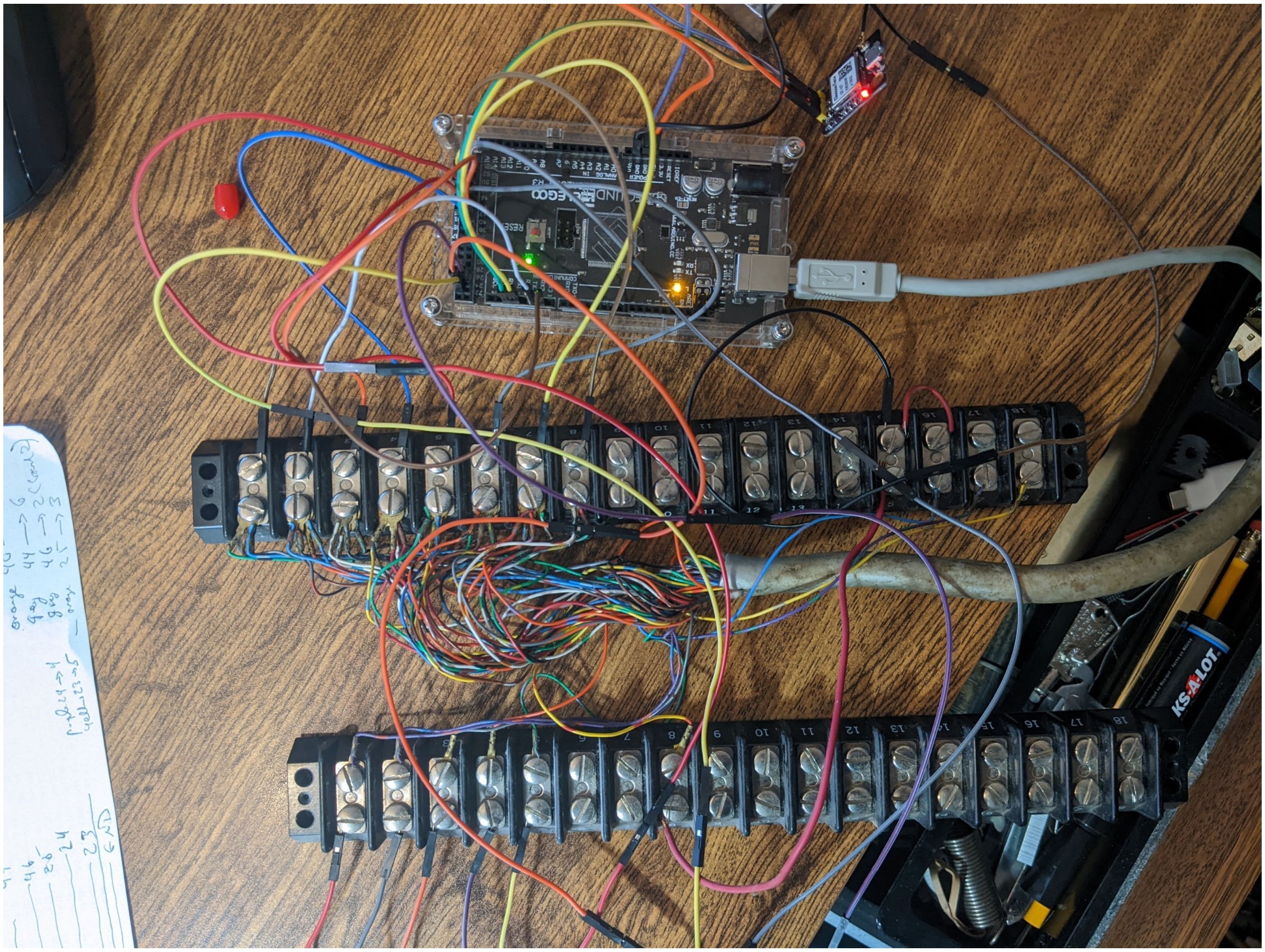
Remote Enable = Pin 42
Ground = Pin 50
All functions are negative true, TTL.
Levels: Low, +0.7V max
High, +2.0-5.0V

Each decimal frequency digit corresponds to 4 BCD input pins on the PTS-160 synthesizer, except that the 10 MHz digit is treated as Hexadecimal, in order to cover frequencies up to 150 MHz with a single set of 4 pins. For PTS-synthesizerss covering frequencies above 160 MHz, the 10 MHz digit uses the same BCD format as the less significant decimal digits, and there is in addition a 100 MHz digit which uses the same format. There are 4 latch pins, each of which (except for the 10 MHz latch pin in the case of the PTS-160 model) is shared by two decimal digits. This makes it possible to reduce the number of Arduino BCD output pins from 9 x 4 = 36 to just 8 pins (plus the latch pins and ground) by paralleling the 0.1 Hz, 10 Hz, 1 kHz 100 kHz, and 10 MHz PTS pins for each BCD digit and similarly paralleling the 1 Hz, 100 Hz, 10 kHz, and 1MHz PTS pins (and also the 100 MHz pin for synthesizers covering frequencies above 160 MHz) for each BCD digit. With this scheme, in order to enter the BCD data for a particular decimal digit into the PTS, the 4 BCD ports on the Arduino that are associated with that decimal digit are set to their appropriate values for that decimal digit and then the latch for that decimal digit is briefly strobed to enter the data into the PTS for that decimal digit. The BCD values for the next decimal digit are then placed into the appropriate 4 BCD ports on the Arduino and the latch for that decimal digit is briefly strobed to enter that data into the PTS. This is repeated for each decimal digit until the PTS has been fully programmed for the given frequency. Note that the PTS uses negative logic, so to provide the BCD digit for 7, for example, the Arduino pins must be set to 0,0,0,1 and NOT 1,1,1,0. The latch is briefly strobed to 1 (and not 0) in order to enter the data into the PTS-160, with the latch signal remaining at zero between these data entry events. Pin 42 of the PTS also needs to be grounded to either pin 50 or pin 21 of the PTS and the this ground needs to be connected to one of the Arduino's GND pins. The appendix contains the details of the connections between the Arduino and the PTS as required by my Arduino sketch code.

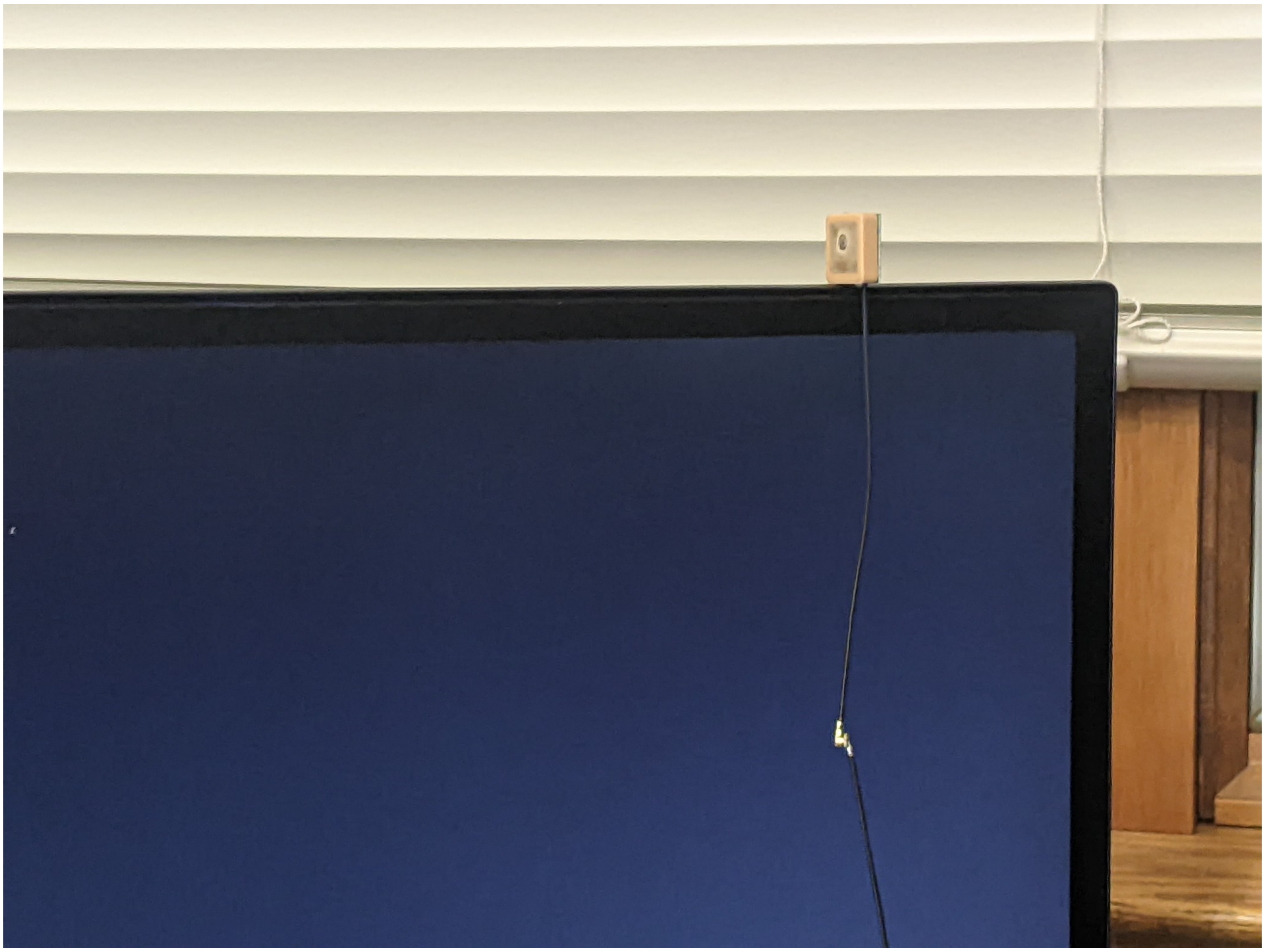
The Arduino has no problem driving the PTS when wired in this fashion, and no buffers or pull-up resistors are required between the Arduino and the PTS-160.

11. I had first used the PTS Synthesizers for amateur radio purposes back in the 1990s when I made them the basis of my first EME receiver, and I had saved the wiring harness that I made up at that time

so for this quick project I just reused that harness. This meant that I didn't need to wire a new 50-pin connector, nor parallel once again the 9 wires for each BCD digit, etc. as all of this had already been done. You can see this wiring as well as the Arduino MEGA2560 that I used for this project in the image below:



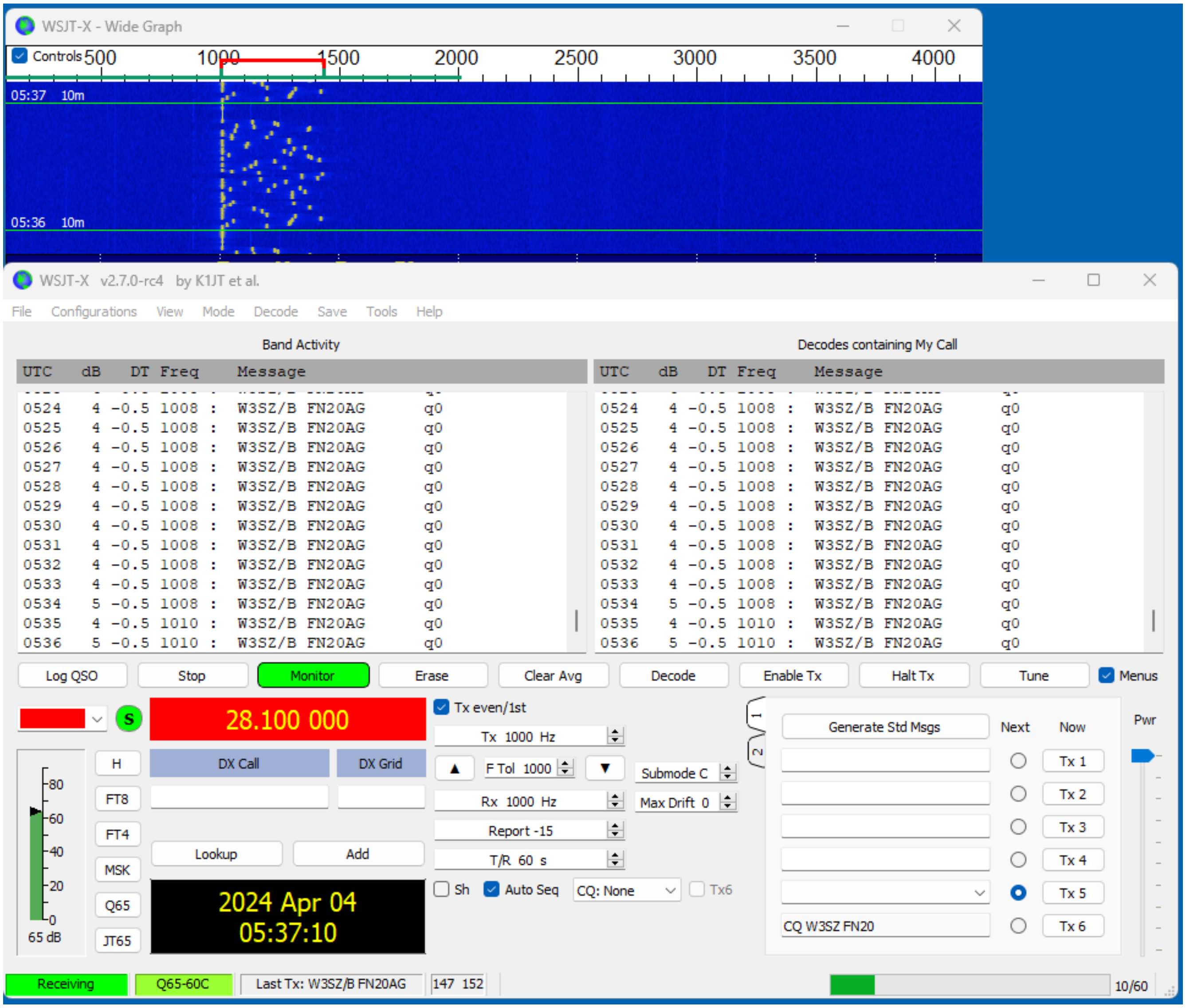
The tiny circuit board with a glowing red LED just above the upper right corner of the Arduino MEGA 2560 is the Goouuu GT-U7 GPS board. This amazing little board gets GPS lock within seconds even with its tiny patch antenna sitting on top of one of my video screens in my shack and hidden from the outside world by mostly closed aluminum Venetian blinds. You can see this tiny antenna, rotated 90 degrees from its proper attitude but still working nicely in the image below:



The image at the top of the next page shows the rear of the PTS-160 with the 50-pin Centronics jack:

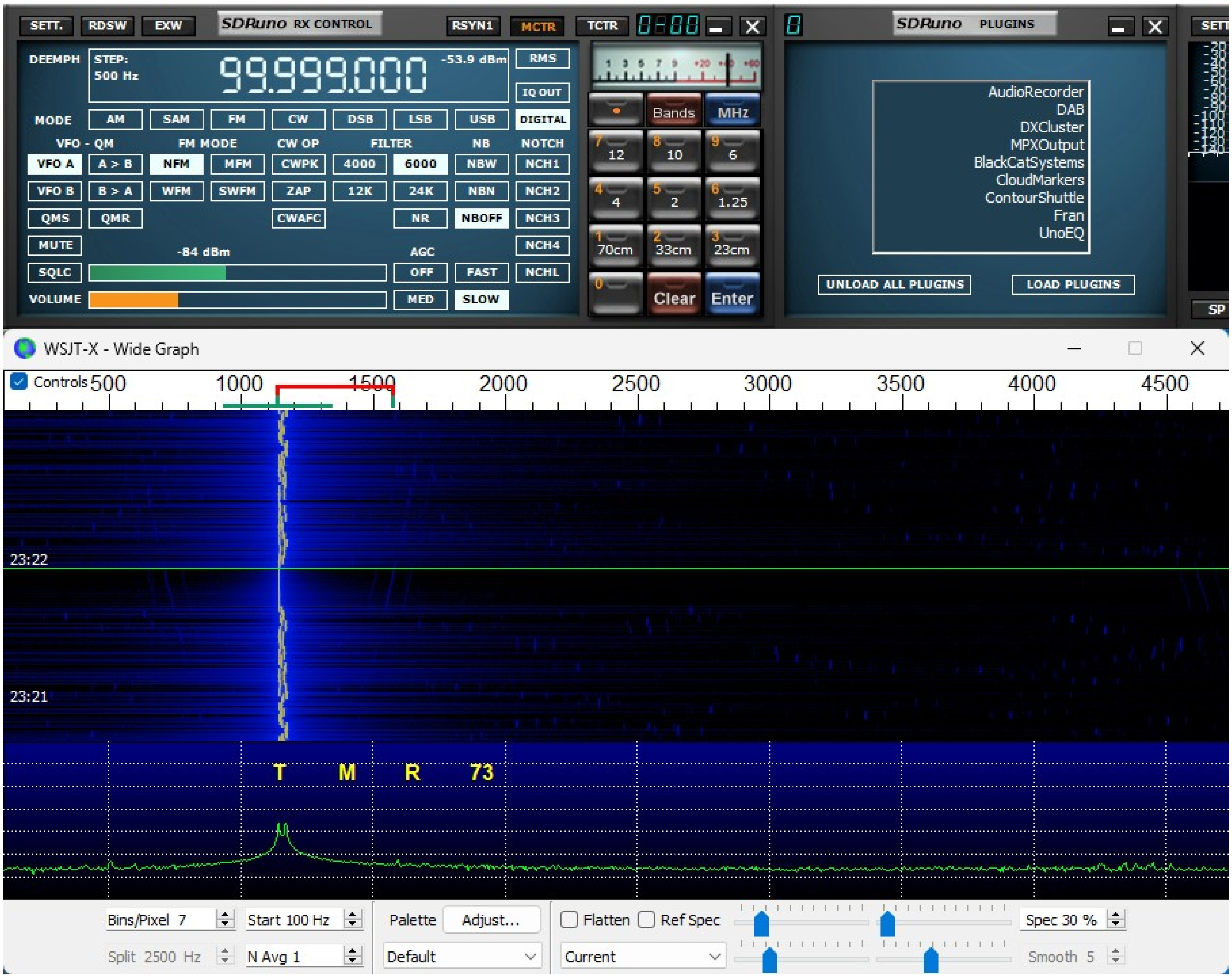
The image shown below is of the front of the PTS. The frequency control knobs are disabled in this application.

The image in the next page shows the PTS160 beacon signal generated by this method using a multiplication factor of 1 as displayed and decoded by WSJT-X:

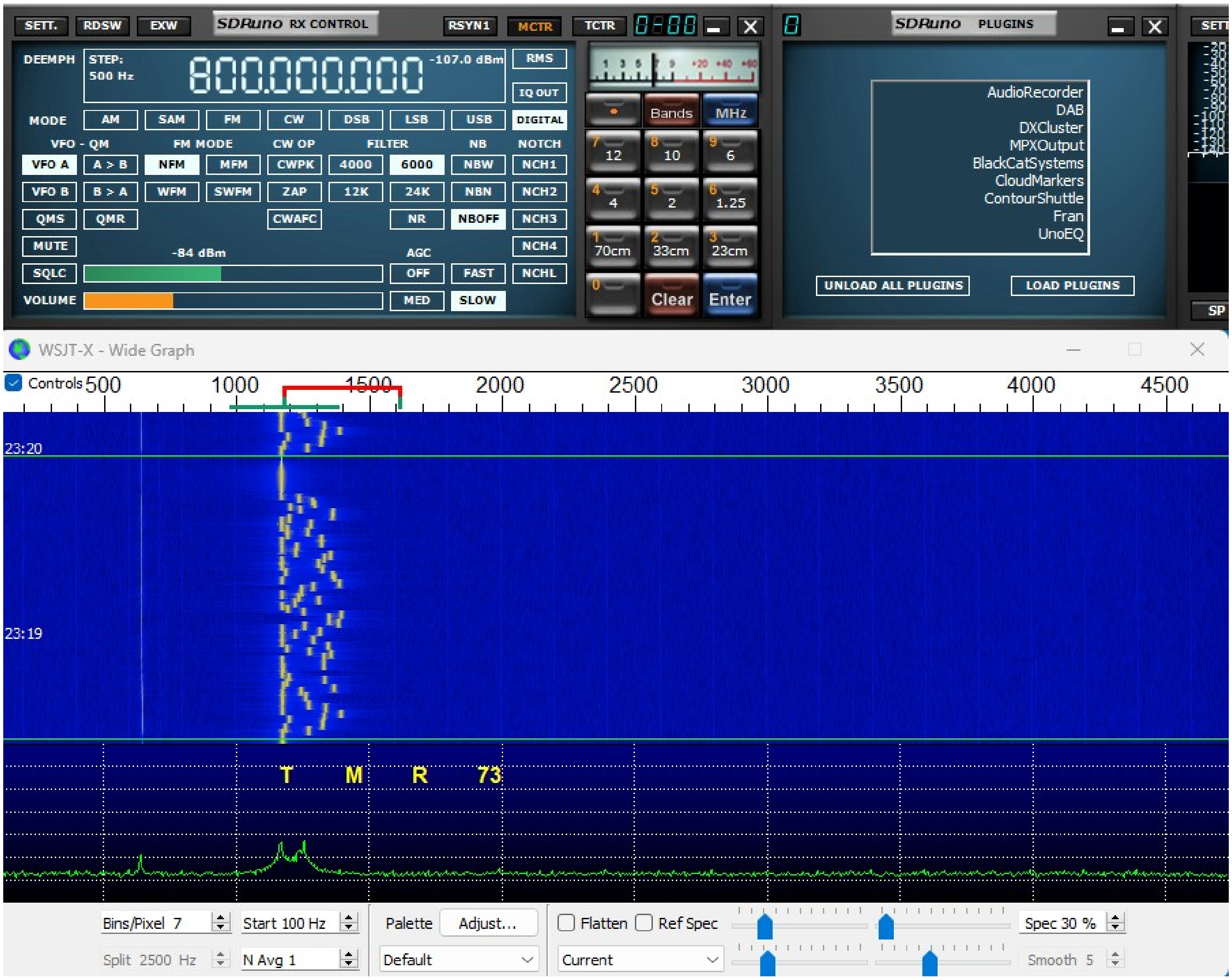


The series of images below show the results when frequency multiplication is used. For this demonstration the PTS-160 described above was used with a base frequency of 100 MHz and a DownEastMicrowave MicroLO board was used to frequency multiply this signal. A frequency list using this base frequency was generated by the method described above using a multiplier of 15 (giving a target frequency of 100 * 15 = 1500 MHz) and loaded into an Arduino MEGA 2560 which was used to control the PTS-160. The signal was received by an SDRPlay RSP1a at the fundamental frequency and at frequencies up to the 15th harmonic, 1500 MHz. Signal levels at higher harmonics were below the threshold for detection. The audio output of the RSP1a was supplied to an instance of WSJT-X 2.7.0-rc4 set to receive Q65-60C signals.

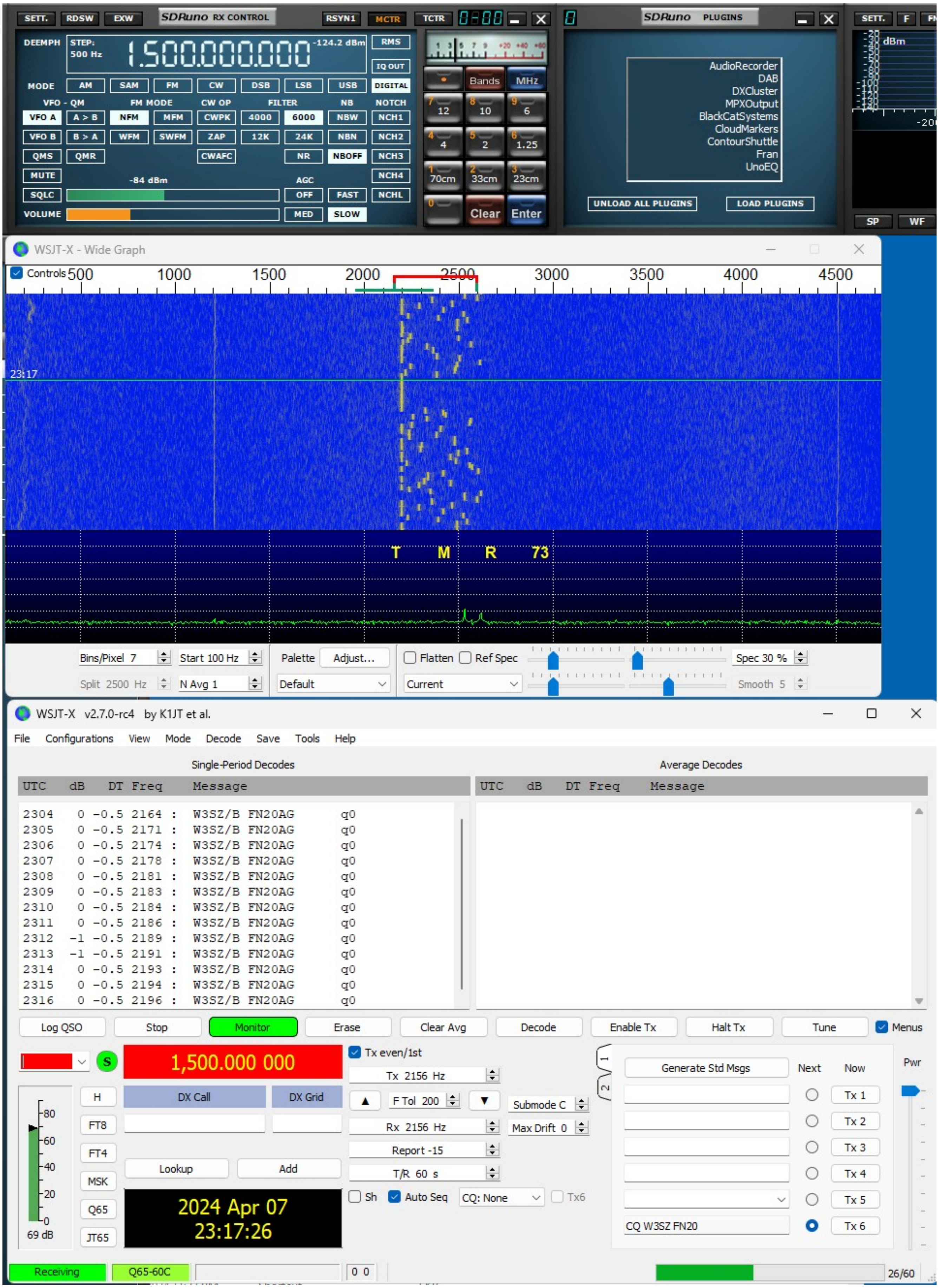
The first image below shows the PTS-generated tones at the fundamental frequency, 100 MHz. You can see that at the fundamental frequency the tone spacing and the total signal bandwidth are much smaller than expected for the Q65-60C mode, the bandwidth of which is indicated by the red bracket extending from approximately 1200 to 1600 Hz. Of course, at the fundamental frequency the spacing and total bandwidth are one fifteenth of their expected values:



The next image below shows the eighth harmonic, at 800 MHz. The tone spacing and total bandwidth are greater than they were at the fundamental frequency, and are now 8/15 (0.53) of their expected value:



The final image, shown below, gives the result at the fifteenth harmonic, 1500 MHz, which was our "target" frequency given that we used a multiplier of 15 when we generated the frequency file. You can see that the tone spacing appears to be as expected for Q65-60C, and that the beacon message consistently decodes correctly when received at 1500 MHz:



SETT.
RDSW
EXW
SDRuno RX CONTROL
RSYN1
MCTR
TCTR
DEEMPH
STEP:
500 Hz
1.500.000.000
-124.2 dBm
RMS
IQ OUT
DIGITAL
MODE
AM
SAM
FM
CW
DSB
LSB
USB
VFO - QM
FM MODE
CW OP
FILTER
NB
NOTCH
VFO A
A > B
NFM
MFM
CWPK
4000
6000
NBW
NCH1
VFO B
B > A
WFM
SWFM
ZAP
12K
24K
NBN
NCH2
QMS
QMR
CWAFC
NR
NBOFF
NCH3
MUTE
-84 dBm
AGC
NCH4
SQLC
OFF
FAST
NCHL
VOLUME
MED
SLOW
Bands
MHz
12
10
6
4
2
1.25
70cm
33cm
23cm
Clear
Enter
SDRuno PLUGINS
AudioRecorder
DAB
DXCluster
MPXOutput
BlackCatSystems
CloudMarkers
ContourShuttle
Fran
UnoEQ
UNLOAD ALL PLUGINS
LOAD PLUGINS
dBm
SP
WF
WSJT-X - Wide Graph
Controls
500
1000
1500
2000
2500
3000
3500
4000
4500
23:17
T
M
R
73
Bins/Pixel 7
Start 100 Hz
Palette
Adjust...
Flatten
Ref Spec
Spec 30 %
Split 2500 Hz
N Avg 1
Default
Current
Smooth 5
WSJT-X v2.7.0-rc4 by K1JT et al.
File
Configurations
View
Mode
Decode
Save
Tools
Help
Single-Period Decodes
Average Decodes
UTC dB DT Freq Message
2304 0 -0.5 2164 : W3SZ/B FN20AG q0
2305 0 -0.5 2171 : W3SZ/B FN20AG q0
2306 0 -0.5 2174 : W3SZ/B FN20AG q0
2307 0 -0.5 2178 : W3SZ/B FN20AG q0
2308 0 -0.5 2181 : W3SZ/B FN20AG q0
2309 0 -0.5 2183 : W3SZ/B FN20AG q0
2310 0 -0.5 2184 : W3SZ/B FN20AG q0
2311 0 -0.5 2186 : W3SZ/B FN20AG q0
2312 -1 -0.5 2189 : W3SZ/B FN20AG q0
2313 -1 -0.5 2191 : W3SZ/B FN20AG q0
2314 0 -0.5 2193 : W3SZ/B FN20AG q0
2315 0 -0.5 2194 : W3SZ/B FN20AG q0
2316 0 -0.5 2196 : W3SZ/B FN20AG q0
Log QSO
Stop
Monitor
Erase
Clear Avg
Decode
Enable Tx
Halt Tx
Tune
Menus
1,500.000 000
Tx even/1st
Tx 2156 Hz
F Tol 200
Submode C
Rx 2156 Hz
Max Drift 0
Report -15
T/R 60 s
Sh
Auto Seq
CQ: None
Tx6
H
FT8
FT4
MSK
Q65
JT65
DX Call
DX Grid
Lookup
Add
2024 Apr 07
23:17:26
69 dB
Generate Std Msgs
Next
Now
Pwr
Tx 1
Tx 2
Tx 3
Tx 4
Tx 5
Tx 6
CQ W3SZ FN20
Receiving
Q65-60C
0 0
26/60

Summary. This project describes the use of a PTS-160 synthesizer as a JT-mode beacon. The synthesizer can either be used without multiplication, or with multiplication (or with multiplication and mixing) to achieve a higher beacon frequency. When multiplication is used, the software described adjusts the spacing of the JT tones so that after multiplication they will have the proper spacing for the chosen JT mode and submode at the actual beacon frequency. The project requires in addition to a PTS synthesizer only an Arduino such as the MEGA2560 used here and a GPS module such as the Goouuu GT-U7 used here. An Arduino MEGA 2560 is currently priced at $17.99 at Amazon, and the GT-U7 is priced at $14.99 there. Depending on the frequency selectivity of the antenna used, some filtering might be needed to attenuate unwanted harmonics.

Both the C# program which generates the frequency files and the Arduino sketch are available on request.

Appendix. Interconnections between the Arduino MEGA 2560 and PTS-160 for the Arduino sketch as written.

Arduino Pin	Centronics Pins
15	15,13,9,5,1
16	16,14,10,6,2
40	40,38,34,30,26
41	41,39,35,31,27
43	11,7,3,17
44	12,8,4,18
49	36,32,28,19
45	37,33,29,20
47	47
46	46
25	25
24	24
23	23
GND	42,21