Forty-Niner PLL – Operating Manual

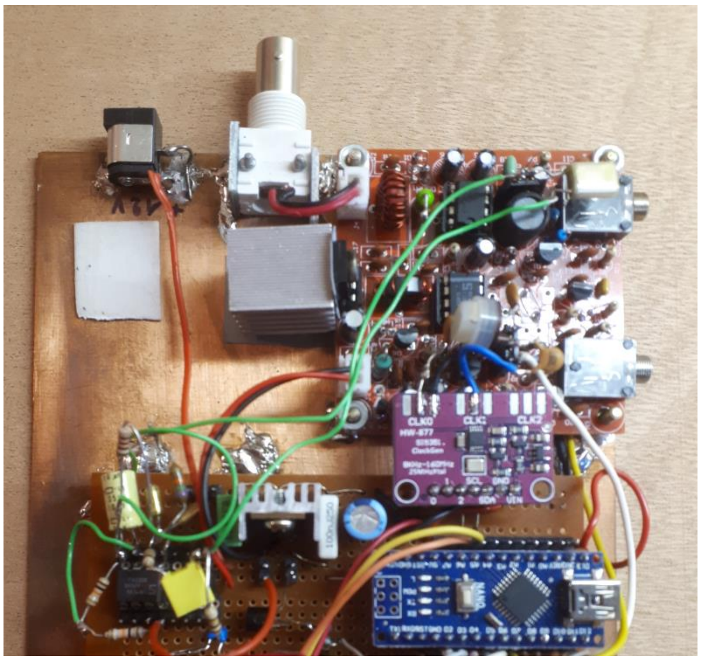
Another 49'er project with a digital VFO.

Hardware : Luc ON7DQ & Gil ONL12523 – Software: Gil ONL12523

Connections

DC 12V ANTENNA

| |

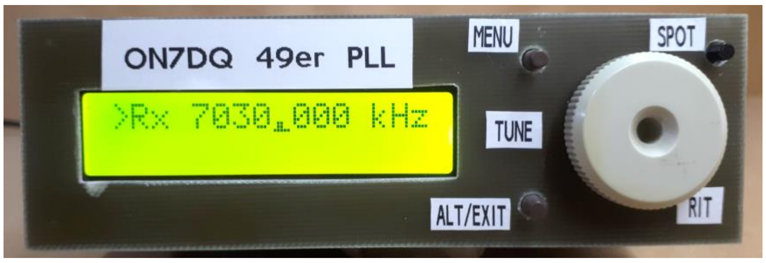


HEADPHONES (or external speaker)

STRAIGHT KEY
3.5mm stereo but only uses TIP & SHIELD

Arduino USB to PC (for programming)

The front panel holds the LCD, rotary encoder with switch, and three more pushbutons.



Using the Buttons

Press = a short press on a button,
Long Press = press a bit longer (or what did you think ? ;-)
Hold = hold the button for as long as needed, depending on the context.

STARTUP functions

EEPROM Erase

Hold the MENU button while powering up or pressing RESET on the Arduino, this will give you the option to erase the EEPROM. It is advised to do this if your Arduino has previously been used for another project, or if there was a major software update and you get a garbled display.
Press MENU to erase the EEPROM, EXIT to cancel and proceed to normal operation.

PLL Calibration

For best results, you should power the rig for half an hour or so, to “warm it up”.
Hold the EXIT button while pressing RESET on the Arduino, this will bring you in a CALIBRATION routine. A 10 MHz signal is sent to CLK2 of the Si5351 PLL. Connect a frequency counter if you have one, or connect a short piece of wire and listen to the signal on a well-calibrated receiver.

If you change your mind, just press EXIT to leave the calibration routine.

To calibrate, press MENU once more, and start adjusting the frequency with the rotary encoder until you have exactly 10 MHz. Press MENU again to store the calibration factor and you will return to normal operation.

[TIP 1: I used my IC7300 and the AUDIO display : tuning your receiver to 10.001 kHz in LSB, then 9999.0 kHz in USB should give a 1 kHz signal in both cases, which is easily seen on the audio spectrum]

[TIP 2: in North America, you can probably use the signal of WWV/WWVH on 10 MHz and try to zero beat with that signal]

MENU operation

Press the MENU button to get into the menu.
Turn the rotary encoder to select one of the menu items.
Press MENU again to adjust the value.

```
7 Beacon
Stored in EEPROM
```

Press MENU again to store the value, a confirmation message will be displayed like in above example.
Press EXIT to leave without storing.

If you have used the MENU button to store a value, you will remain in menu mode and can select another menu item. With EXIT you will immediately leave the menu and return to normal operation.

BAND

```
1 BAND
   40m
```

Band selection. Available bands are now : 160-80-75-60-40-30-20m and ALL.
Of course you will need to change the low pass filter and the series LC bandpass filter for the appropriate band, this is left to your construction skills. The default band is 40m.
ALL = frequency range from 10 kHz tot 100 MHz, in case you want to use the PLL module as a simple RF generator for experiments. The band table can be changed in the Arduino main program (look for array **`band_info band[]`**).
If you have previously selected a band and stored it, it will show up as the first option when you return to this menu item.

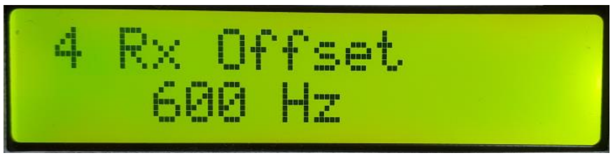
FREQ Minimum and FREQ Maximum

```
2 FREQ Minimum
    6000.000 kHz
```

```
3 FREQ Maximum
    7200.000 kHz
```

Selection of the lower and upper band limits.
When selecting a band, the standard band limits for Region 1 are set, but here you can set whatever band limit you want. In the example above, I selected 6 MHz as a lower limit, to be able to listen to the 49m broadcast band (tuning zero beat in SSB to an AM station is a bit tricky but it works).
The default band limits can also be changed in a table in the Arduino code (look for the array **`band_info band[]`**).

Rx Offset

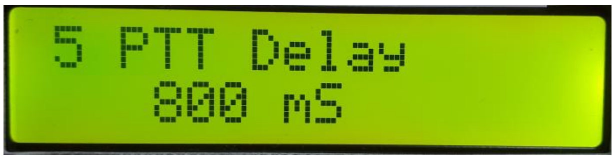

4 Rx Offset
600 Hz

Select the desired frequency offset. It can be set between -990 and + 990 Hz.

[NOTE : a positive offset would be the LSB mode, since the oscillator is above the signal, a negative offset is then the USB mode. If this is more confusing than helpful, disregard this note ;-)]

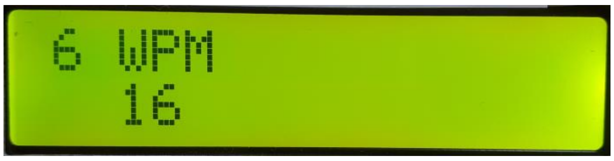
The sidetone will be set at the same frequency (in absolute value of course).
To check the tone, just press the key.
This could also be used as a simple **code practice oscillator**, since only the tone is generated, without transmitting. Try it !

PTT Delay


5 PTT Delay
800 mS

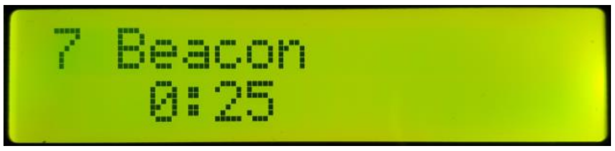
Here you can set the PTT delay time from 10 ms to 10 seconds.
Setting the delay at 10 ms will give a smooth **Full-QSK** operation.

WPM


6 WPM
16

Set the CW speed for the built-in CQ or beacon messages, from 5 to 49 WPM.

BEACON Mode


7 Beacon
0:25

When you want to use the 49'er as a beacon, or maybe as a foxhunt transmitter, or just for repeated CQ calls, you can set a beacon repeat time here. Default is OFF.
Time can be set from 1 second to 90 minutes, and is shown in **MM:SS** format.
The time is measured from message start to message start, so your time should always be greater than the time to send one message, depending on the WPM speed set in menu 6.
Select the message as described in the message operations further on.

Normal Transceiver Operation

Note: the last tuned frequency, the amount of RIT and the status of SPLIT will be saved to EEPROM after 5 seconds idle time. Also what CQ/BEACON message was in use will be saved.

The idle time can be changed in the source code, look for the line

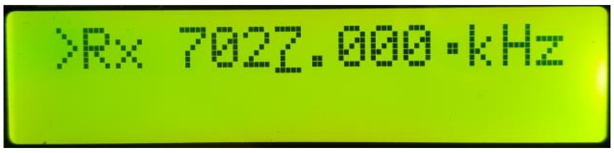
```
byte saveTimer = 5;
```

Receiving

Volume control

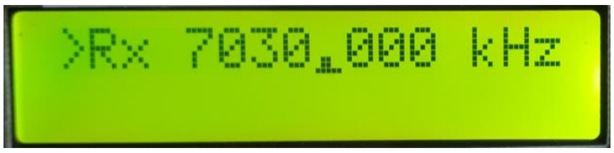
Sorry … there is no volume control nor RF gain control in the rig (yet).
This is left to your ingenuity, and should not be too difficult.
One easy solution is to get a computer headset with built-in volume control, or use an amplified (PC) speaker which most often has a volume control of its own.

TUNING



Use the rotary encoder to tune the band. Simple.

The tuning STEP can be changed: HOLD the encoder button and at the same time rotate the encoder slowly left or right, the UNDERLINE cursor will show what step you have selected.
In the above example we have selected a 1 kHz step.



This is a special case: If the cursor is under the dot, the step is **500 Hz.**

At the band edges, you cannot tune any further (to prevent transmitting out of band).
Unless you have changed the band limits via the menu of course ;-)

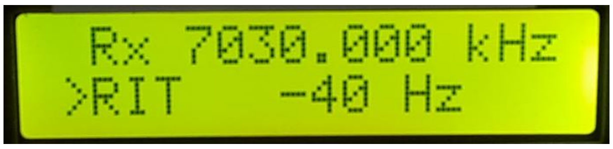
NOTE:
This is a direct conversion receiver, and you will hear a station twice, at each side of the oscillator. When you have selected a positive OFFSET (e.g. 600 Hz), you need to tune a station with a raising tone for an increasing tuning frequency. If you're tuning UP and the tone decreases, you're tuning the station on the "wrong" side of the oscillator. With a negative offset, it works the other way round.

SPOT

To help you in tuning the station ‘zero beat’, press the SPOT button.
The sidetone will be injected at a lower volume and you can try to get the station as close as possible to the tone. The tuning step will automatically be reduced to 10 Hz, and will return to the chosen step after releasing the SPOT button.
The volume of the SPOT tone can be set in the file DDSGen.ino , look for the line with
#define SPOT_VOLUME 80
Values are from 0 to 240

Another use of the SPOT function is to quickly “fine tune” the frequency: hold the SPOT button and tune in 10 Hz steps, release the button and you’re back at the (coarser) step you had before.

RIT (Receiver Independent Tuning)

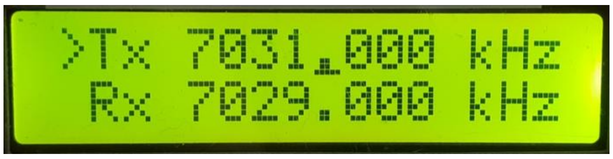


```
  Rx 7030.000 kHz
 >RIT    -40 Hz
```

Press the encoder button and the RIT offset will show on the second line of the display.
You can tune in 10 Hz steps , up to +/- 10 kHz.

Press the encoder again to turn off the RIT, but the RIT value will be retained for the next use.
To reset RIT to zero, press the EXIT button while RIT is ON.

SPLIT



```
 >Tx 7031.000 kHz
  Rx 7029.000 kHz
```

Holding the menu button will toggle SPLIT ON or OFF.
Two frequencies will be shown:
On the top line is the Tx frequency, on the bottom line is the Rx frequency.

Which one you will tune is indicated by a “**>**”
You can alternate between the two with a press of the ALT/EXIT button.
When SPLIT is turned off, the simplex frequency is set to the last used Rx frequency.
In the example above you will return to 7029.000 simplex.

Transmitting

Not much to say here … just hit the key and enjoy the QSO !
Or use one of your pre-programmed messages as described below.

Built-in Messages

Holding the ALT/EXIT key and at the same time rotating the encoder selects between several pre-programmed messages.
The messages have to be entered on beforehand in the Arduino code, look for this array, e.g.:

```
const char* morseText[] = { "CQ CQ DE ON7DQ ON7DQ PSE K",
                            "TEST RBN DE ON7DQ ON7DQ TEST RBN",
                            "VVV DE ON7DQ/B ON7DQ/B JO11KF PWR 1 WATT = 73",
                            "THE QUICK BROWN FOX JUMPS OVER THE LAZY DOGS BACK",
                            };
```

You can add as many messages as you like, until the Arduino memory is full !
See Appendix 1 for a list of all possible Morse characters and prosigns that can be used here.

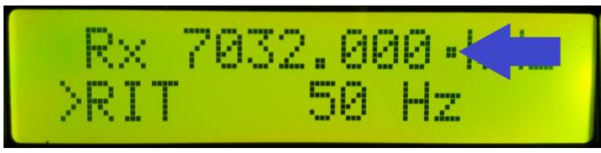
On release of the ALT/EXIT button the message is transmitted at the WPM speed set in menu 6.
Press the ALT/EXIT button to interrupt transmission.
Long press of the ALT/EXIT button will send the selected message again.
You can also automatically repeat your CQ or other message, see Beacon operation below.

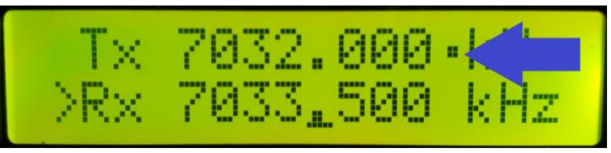
BEACON Operation

If you have set a beacon time in menu 7, you can start beaconing with a long press on the ALT/EXIT button. This can also be used for repeated CQ'ing, or maybe for a simple foxhunt transmitter.

Transmission is immediately interrupted with a press on the ALT/EXIT button, but after the set time, transmission will be resumed. A countdown timer will show when the next transmission will begin.
If you want to stop beaconing completely, press the MENU button.
The beacon time is not changed however, you can start another beaconing session by long press on the ALT/EXIT button.

If you are not in RIT or SPLIT mode, the beacon operation will be obvious : you will hear the code being sent, see the text scroll by on the display, and during the idle time, the countdown timer is shown on the display.
But what if you want to send a repeating CQ or beacon message when using RIT or SPLIT ?
There is no more room on the display to show the text or the countdown timer.





indicator in RIT mode

indicator in SPLIT mode

For that purpose , an **indicator** is shown between the frequency and the kHz on the top line of the display. So you will always know that the beacon time (menu 7) is not in the OFF position.

The symbol for this indicator can be changed in the source code, look for this line in the main program:
byte charB[8] = {0x0, 0x0, 0x0, 0x6, 0x6, 0x0, 0x0, 0x0}; // custom beacon indicator, make your own

The symbol is made in an 8x5 matrix, with a 1 for an active pixel. The example above makes a small square. Convert the binary values to hex, then enter them in the character array above.
00000 = 0x0
00000 = 0x0
00000 = 0x0
00**11**0 = 0x6 // 2 pixels are ON
00**11**0 = 0x6 // 2 pixels are ON
00000 = 0x0
00000 = 0x0
00000 = 0x0

Keyer ?

There is no keyer in our software (yet). If you really must use electronic keying, you can easily build the K3NG keyer with another Arduino Nano (see https://github.com/k3ng/k3ng_cw_keyer).

Or why not use our **OST Morse Box** ?
It's another fun building project, it has a built-in touch paddle, iambic A& B keyer, multiple message memories, an optional decoder and a lot of practice tools.
It has a keying output via an optocoupler, ideal to use with the 49'er !
see https://github.com/on7dq/OST-Morse-Box (basic version)
and
https://github.com/on7dq/OST-Morse-Box-DG (extended version)
and
https://github.com/on7dq/OST-Morse-Box-V3 (optional Windows Control program)

Or watch this series of videos on my YouTube channel https://youtu.be/ROU9pVeI86k

Have fun with the Forty Niner PLL!

Luc ON7DQ and Gil ONL12523

January 2023

Appendix 1 : Morse characters

Character set

The following characters can be entered in the messages:

All letters A – Z (upper case or lower case)

All numbers 0 – 9

Punctuation : ; “ & ‘) , - . / ? @ [^ _

The following symbols are used to send the corresponding prosign:

$ or = <BT>

% or] <SK>

(<KN>

+ <AR>

How is the morse code created ?

All characters are kept in a table in the Morse.ino file.

```
const unsigned char morsewoord[] PROGMEM = {
          0x00,0x00,0x92,0xCA,0xD1,0x85,0xC2,0x9E,  // Prosigns
          0xD6,0xAD,0x00,0xCA,0xB3,0xA1,0x95,0xD2,
          0xDF,0xCF,0xC7,0xC3,0xC1,0xC0,0xD0,0xD8,  // 0
          0xDC,0xDE,0xB8,0xC8,0x00,0xD1,0x00,0x8C,
          0xC4,0xF9,0xE8,0xEA,0xF4,0xFC,0xE2,0xF6,  // A
          0xE0,0xF8,0xE7,0xF5,0xE4,0xFB,0xFA,0xF7,
          0xE6,0xED,0xF2,0xF0,0xFD,0xF1,0xE1,0xF3,
          0xE9,0xEB,0xEC,0xD5,0x00,0x85,0x00,0x00,
         };
```

Each character in a message is converted into its ASCII code, and with an offset of 32, its position in the table is determined.

The HEX numbers you see above are to be interpreted as binary numbers.
Then, when looking at the byte from left to right, we might encounter ones or zeroes.
It doesn’t matter how many ones you see, but the first zero that is found is a START marker.
All bits that come after that are to be transmitted, where 0 means a DOT, and 1 means a DASH.
This means we can only store Morse codes for symbols up to 7 ‘elements’ (dots ot dashes).
e.g. We cannot store the ‘error’ symbol of 8 dots, but that is not a big deal, since most hams don’t use it and send three or four dots at a lower speed (sounds more like 3 or 4 E’s in a row).

An example will make it clear.
The letter A has ASCII code 65, minus 32 makes 33, in the table this is the **34th** position (we count from zero), where we find HEX **0xF9**, or binary ~~11111~~**001**
The first 5 ones are skipped, the zero (in RED) is the start sign, so all we have to send is a zero and a one, so this becomes DIT DAH = indeed an ‘A’ in Morse code.