Simple JT4 Code Generator

Andy Talbot G4JNT August 2011

JT4 Overview

JT4 in all its variants (A-G) consists of a four tone Multi Frequency Shift Keyed (4-MFSK) waveform, with the spacing between the tones chosen depending on the frequency band and expected spreading. [1] The MFSK message consists of 207 symbols (one of four sequential tones) transmitted at a rate of 4.375Hz, the whole message therefore taking about 48 seconds to send. A rigid timing structure is in use, and the start of the transmission must coincide with the UTC minute interval. For beacon usage, the even minute has been universally chosen as the reference start time for beacons using this mode. However, the decoding software does have a monitor function whereby transmissions in both even and odd minute slots are decoded.

For the decoder to work correctly, the start point must be accurately defined, being no more than a few seconds late, and no more than one second early (the protocol was originally designed for EME with its 2 seconds delay). The entire message contains exactly 13 characters taken from an alphabet of letters, numbers and a few punctuation symbols. More details of JT4 coding can be found at [2]

Code Generator Module

The unit described will generate correctly timed and formatted JT4 symbols on a two-bit parallel interface – the code updates synchronously at the 4.375Hz symbol rate; no additional clock is provided. The code appearing on the interface, the binary value from 0 to 3 corresponds to the individual tone number of the JT4 transmission and is designed to be connected to a four-frequency generator such as that in [3]

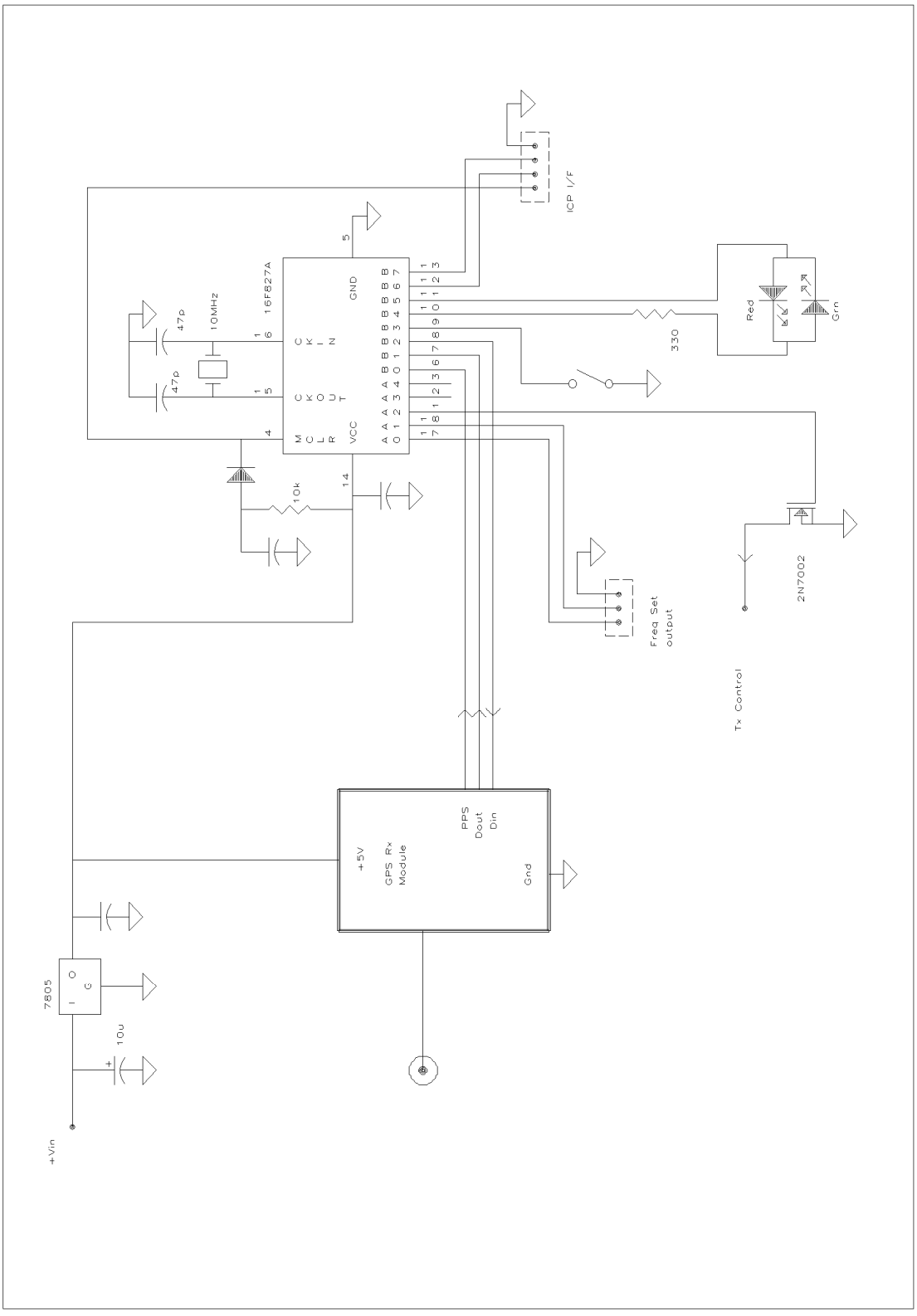
A 16F627 (or 16F628) PIC monitors the GPS serial data line and decodes the real time information from the GPS. Every even minute, or every minute depending on requirements, the 00 seconds marker is identified and 207 pre-stored symbols are sequentially output on the two 0/5V logic level outputs. Immediately the end of the transmission, an optional CW message is keyed via an open drain FET allowing additional identification. The key line is active (FET on) during the JT4 transmission.

Annex A contains details of a Free Running version of the software that does not need a GPS module, and relies only on the PICs internal oscillator for timing.

Connecting the GPS module.

The description that follows, as well as the PIC firmware supplied, assumes that serial data in one of two formats is available. The proprietary binary format given by the Motorola Oncore or M12 type GPS module at 9600 baud or standard NMEA text messages at 4800 baud carrying the *$GPRMC* string. The polarity of the data can be selected at the time the PIC firmware is compiled. Either native 5V logic or RS232 polarity can be catered-for

Figure 1 shows the circuit diagram of the generator module. Two input lines carry the serial data and 1 pulse-per-second synchronising signal. There is also a third GPS interface connection shown, an output from the PIC to the GPS receiver. At the moment this is not used and does not have to be connected. It has been included for any future version that could include GPS receiver initialisation. A red-green LED shows the operating status and indicates whether the GPS is synchronised. When valid data appears from the GPS module long green flashes are shown. When the GPS receiver is not synchronised, these change to short flashes. When JT4 data is being sent, the LED flashes red at half-symbol rate, and then shows the CW data.



PIC Coding Details

All information relating to the message, frequencies and setup need to be programmed into the PIC at the start. There is no facility for field updating with an RS232 interface and all values need to be included within the source file which is compiled to give the .HEX file for download to the PIC device. Compile-time flags are used to define the data polarity and format from the GPS.

The basic PIC firmware is contained in the source file *JT4GEN2.ASM* .The JT4 symbol information resides in an auxiliary include file *JT4SYMBS.INC* which can be generated automatically by the utility *GENJT4.EXE*. Alternatively, the symbols can be derived from the WSJT software, following Joe's instructions supplied with the software suite, formatted and entered manually into the include file. Each of the 207 symbols is formed from two bits giving a value from 0 – 3 which are packed four to a byte, most significant first to give 51 bytes in total. (As listed, they are read out in order left to right, top to bottom)

Change the compile-time flags and CW message data to suit your requirements, and generate a new JT4SYMBS.INC file. Save the new assembler file and use a utility such as MPASM (available from the Microchip website or included within the MPLAB suite) to generate a new .HEX file for programming into the PIC device

The code supplied is designed for 16F627 and 16F628 type devices.

Compile-time flags.

These appear at the start of the assembler listing as shown in the table below

NMEAPol defines if the polarity of the data coming from the GPS receiver is 0/5V logic level as supplied directly by most GPS modules, or RS232 polarity for direct connection to a PC. Some early Garmin modules supply this latter polarity, as do some GPS receiver systems. . Use **0** for 5V Logic level / polarity, **1** for RS232.

Please note that if true positive/negative RS232 voltage levels are encountered, an additional resistor of around 4k7 needs to be inserted in the Data In line to prevent excessive current into the PIC interface pin

GPSDataType should be set to **0** for Motorola binary format data at 9600 baud; Use **1** for NMEA ASCII format at 4800 baud

BOTHMINUTES defines if the JT4 is sent every minute, or every two minutes on the even minute boundary. It should be set to **0** for convention even minute transmissions, and set to **1** for near 100% duty cycle transmission of the JT4 message every minute. There is no option for transmitting only on the odd minutes.

IGNOREPPS allows timing information to be derived from the GPS data stream alone, without any need for the 1-PPS signal. This simplifies the connection for some GPS receiver modules, but does mean the transmission timing could have up to one second uncertainty. Set to **0** for normal high accuracy timing using the 1-PPS signal, Use **1** for GPS serial data based timing only

Compiler Constants

CWSPEED is a compiler constant and defines the dot length of the CW, in milliseconds. Use **d'100'** for 12WPM, **d'75'** for 16WPM etc.

BAUD9600 and BAUD4800 should not be changed.

```
GPSDataType          =       1       ;1 = NMEA  0 = Motorola Binary
NMEAPol              =       1       ;1 = RS232 Levels, 0 = TTL
BOTHMINUTES          =       1       ;1 = every minute, 0 = Alternate (even) minutes

CWSPEED    =  d'50'        ;CW Dot length, ms
BAUD9600   =       d'40'   ;6.N + 18 = Fc/Baud  or  N ~ (Fxtal/Baud)/24 - 3
BAUD4800   =       d'83'
```

JT4SYMBS.INC

This include file is generated automatically in exactly the form shown as a result of running the utility ***GENJT4.EXE***. [3] It should not be necessary to alter the file in any way. As the file is regenerated and overwritten each time ***GENJT4*** is run, it is advisable to save a copy under a different name – eg*, GB3SCS_JT4SYMBS.INC*.

The WSJT software does offer the ability to generate the symbol data in a listed form, and users may want to use this route instead – for example to include a 'QSO-type' message into the beacon data instead of 13 characters of plain text. In this case, the individual symbol data in the form of 207 numbers with values 0 – 3 will have to be assembled manually into the EE data bytes, four-at-a-time starting with the most significant pair of bits. For example, if the first eight symbols generated are 3,1,2,0,2,1,3,0, the resulting first two bytes will be b'11011000' and b'10011100' or in hex 0xD8, 0xC0. Both these formats, binary or hex, (or even decimal as d'nn') are acceptable to the compiler. Read the WSJT documentation for further details of how to generate the symbol list.

```
;  JT4 Symbols  generated from GENJT4     G4JNT Jul 2009
;  Message data 'GB3SCS IO80UU'

    de  0x00, 0xD8, 0x14, 0xDA, 0xC4, 0x02, 0x8D, 0x28
    de  0xAA, 0x0A, 0xC7, 0x9C, 0xEF, 0xD6, 0x68, 0xC3
    de  0xA5, 0x74, 0x2C, 0x6A, 0x75, 0x1E, 0xB8, 0x34
    de  0xC4, 0xC6, 0xF5, 0xC4, 0x67, 0x33, 0x9D, 0xA4
    de  0x59, 0x76, 0xA9, 0x65, 0x83, 0x53, 0x73, 0x50
    de  0xC0, 0x51, 0xE9, 0x2B, 0x57, 0x63, 0xE2, 0x34
    de  0x26, 0x73, 0xD6, 0x6C
```

CW Message

At the end of the assembly listing, identify the block of code shown in Table 3. Change the text within the inverted commas, after the Label 'CWMsg' to suit your requirements. A blank string is allowed. Please note, the '0' at the end of this string is essential, and must not be inside the inverted commas. Without it the software will lock up. Also, the Label CWMsg must appear on the extreme left hand side of the listing.

TABLE 3

```
;================================

 org   0x2100

JT4MsgData

  include "jt4symbs.inc"                ;Left Justified JT4 tones, 4 per byte

CWMsg         de     "G4JNT",0

;--------------------------------
```

.**Test Mode**

The link or switch installed on B3 allows continuous carrier at the reference tone frequency for test purposes with key-down. If activated during the CW or JT4 sequence, this is allowed to complete before Test Mode is entered. The red LED is on continuously.

ANNEX A

Free Running Version

The PIC Code ***JT4GEN_FreeRun*** (included in the archive) does not need any external timing information and uses the PIC's own oscillator to generate the appropriate intervals. All connections to the PIC are the same as for the standard version, except that ports B0-B2 are not used. Provisions has been built into the software for use – with care – of crystals of other frequencies. The timing for both the JT4 sequence and the real time count of seconds has to be derived by dividing down from a regular interrupt, which itself is generated by dividing down from the PIC internal clock. Only certain frequencies will meet the requirements for this,; the seconds count has to be exact, and the JT4 timing has to be close enough that at the end of the 48 second sequence, the timing has drifted by no more than a fraction of a symbol.

A spreadsheet ***WSJT_PIC_Timings.XLS*** has been included to assist in calculating the values.

The source code is customised in the same way as for JT4GEN , except that now only the ***BOTHMINUTES*** user flag is available. If crystal frequencies other than 10MHz are to be used, the divider values have to be changed at the start of the listing. See table A1 for calculation of these.

To synchronise timing, the module must be powered up or reset as close to the 00 seconds point as possible. Timing has been arranged so that it launches into the JT4 code immediately after power up / reset. When using the alternative minutes option, odd or even slots are possible depending on the reset point.

The LED flashes red at half symbol rate while sending the JT4 codes and for the CW message; it remains green during the idle period.

Timing accuracy depends on the PIC oscillator remaining on frequency. Assuming a reasonable 10 parts-per-million accuracy, this equates to a drift of one second per 100000s or less than 1s per day. If the frequency drifts low and the clock slows, the resulting permissible timing for successful decoding using the WSJT software can be up to 6 seconds late. It is not so forgiving of early timing, and only 1 second fast is permissible. So if you are unsure of timing drift, or long transmission periods are anticipated, it is safer to start / reset a few seconds after the minute marker.

For long term usage, a TCXO with 1ppm stability should give acceptable timing for several weeks or even months of operation

TMR0 Programmed value

Table A1 Selecting divider values for arbitrary oscillator frequencies

Choose a crystal frequency and enter this into the box under 'PIC Oscillator' in the ***WSJT_PIC_Timings*** spreadsheet. Do not change the values in 'Prescalar, and 'Interrupt Overhead' (4 and 12 respectively). These cannot be altered in this PIC code.

Check the timings are OK against JT4 and "1s timing". If an unsuitable crystal frequency is chosen, "**X no**" will appear in red against the associated entry. If this happens, choose another crystal frequency, or with care, adjust the "Interrupt Division" value. Do not use any value less than 100 in here; values from approximately 100 to 240 are acceptable.

When a satisfactory set of values are found, note the values generated for "TMR0 Programmed value", and the "Int Div N" for JT4 and 1s timing. Also the value for MSDELAY.

Copy these values into the assemble file listing and assemble the code.

How the timing is generated:

The oscillator frequency is divided by 16 (fixed) and clocks the 8 bit TMR0 counter. When this overflows an interrupt is generated. By pre-loading with a constant, ***INTDIVIDER*** each time it overflows, the resulting interrupt rate can be modified. The Interrupt frequency is therefore :Fxtal / 16 / (256 – N') where N' is the value preloaded each time it overflows. There is a bit of a snag as some clock cycles are taken up servicing the interrupt before N' is loaded, so the value has to be modified. The end result is to generate an interrupt at a precise rate – shown as the values against "1s timing"

Two separate counters now count each interrupt. The seconds counter is checked against a value ***INTSPERSECOND*** that is numerically equal to the interrupt rate in Hz, or the "1s Timing" value. When this value is reached it is reset, and the seconds / minutes updated. All are started off at zero when the PIC is switched on / reset.

The Symbol timing is checked in the same way against a value ***JTDIVIDER*** that results in a rate acceptably close to 4.375Hz to give insignificant symbol overrun at the end of the transmission period.

MSDELAY generates the basic 1ms delay used for CW symbols and a couple of other non-critical delay sections.

References

[1] http://physics.princeton.edu/pulsar/K1JT/

[2] http://www.g4jnt.com/JTModesBcns.htm

[3] **4FREQ_PIC_DDS_Source.pdf**