ICARC Fox Transmitters

William Robison

April 12, 2024

This is the start of a manual for the ICARC FOX Transmitters. It covers the 102-73161-25 board as well as the 102-73181 series of boards.
The 102-73161-7 and 102-73161-12 boards are **not** expected to use this software.

It is a work-in-progress right now so suggestion for updates may be sent to **kc0jfq@n952.ooguy.com**.

Full size documents may be found here: http://n952.ooguy.com/eagle

http://n952.ooguy.com//home/wtr/eagle/JUICE/A102_73181/FOX_ICARC.tex

Contents

List of Figures

List of Tables

1 Glossary of Terms

There is an attempt being made to use some terms in this document in a precise manner. Some of the discussions become a bit muddled when terms are used casually.

1.1 FLASH

A non-volatile memory with a slow write cycle. This memory can be erased by command, but the erase cycle and write cycle are much slower than the read cycle.

Update is handled by erasing the entire device.

1.2 ISR

Interrupt Service Routine.

This is a block of code that deals with an even that is not triggered by the normal flow of instructions in the processor.
Examples would be incoming serial traffic or a timer event.

1.3 FRAM

Ferro Magnetic **R**andomn **A**ccess **M**emory.

A type of non-volatile memory that exhibits symmetric access speed. IN other words the write speed is the same as the read speed.

Update is handled a record at a time or by erasing the entire device.

1.4 Processor

This refers to the zNEO system-on-chip. It has an instruction execution engine (i.e. the CPU), program memory, data memory, and a variety of peripherals.

1.5 Sequence

We will use the term **SEQUENCE** in this document to a set of instructions that executed as a group.

Typically this *sequence* is stored in external memory.

1.6 TOY Clock

Time-of-Year clock. A battery backed clock that keeps time when the transmitter is not powered.

1.7 xxx

xxx

2 Motivation

Why all the improvements?

Because we can! And because if FUN!

Most of the shortcomings of the 102-73161 series boards are addressed in this revision. The clock synthesizers used in the 102-73161 boards are at end-of-life and will become more difficult to obtain. Moving to a newer synthesizer hopes to extend the useful life of the design.

In addition, there are some additional control signals presented to the RF daughter board to allow the use of an integrated transceiver module. This transceiver module entirely bypasses the clock synthesizer.

2.1 Requirements

These are required for operation and remain unchanged from the 102-73161 series boards.

2.1.1 2M Band

We are using 2M handheld transceivers as a primary detector.

We must stay within our licensed band.

Transmitter must be able to supply a signal that can be decoded by a normal handheld transceiver.

2.1.2 Battery Operation

We must be able to carry multiple FOX Transmitters, so battery weight may be an issue. Target is a 9V battery or a 6-cell pack.

The switch-mode regulator allows for the use of a much wider range of battery packs without sacrificing conversion efficiency. The battery pack may be anywhere from about 6 1/2 to 7 volts all the way up to 24 volts.

A low power RF power amplifier to extend battery live.

2.1.3 Code Storage

We must store enough bits to identify the transmitter.

We would also like to have enough storage for the message traffic to change during the hunt.

2.2 Desirements

These are desired features

2.2.1 Multiple frequency

We would like to be able to operate the transmitter on more than one frequency.

2.2.2 Multiple band

We would like to be able to operate the transmitter on more than one band. It is probably unreasonable to be able to do this without changes to the output filter unless the filter is moved to the amplifier daughter card.

Implementation:
ICS525 (or ICS307 on the 73181-0 board) clock synthesizer controlled by the zNEO processor. Any frequency that can be generated by the ICS525 (or ICS307) given its clock input may be selected. ICS307 clock synthesizer controlled by the Raspberry PI-Zero. Control bits are supplied through a serial interface, so programming the control word is mandatory with this synthesizer.

The 102-73181-5 and later boards use a Skyworks SI5351 synthesizer, similar to the ICS307, to operate up into the 2M band.

2.2.3 Large CW tables

We get bored with the same old message over and over.

We would also like to be able to operate at any word rate.

Implementation:
Message traffic stored in FRAM, which may be as small as 64Kb (8KB). Most of this memory can be used to store message traffic.
Larger devices, of course, increase the size of the message traffic that may be stored.
CW chipping rate can be selected from 1-WPM to 50-WPM.

The Raspberry PI-Zero made use of a micro-SD card, so storage space was not an issue for that hardware.

The 102-73181 board adds a second serial FLASH position for storing waveform data.

2.2.4 Voice Storage

We must provide sufficient external storage for waveform data. Something on the order of 60 seconds of voice data sampled at a 4KHz rate.

In the 102-73181 design, as noted above in section 2.2.3, a separate storage device is used to store audio data. This separate device is a large low cost flash device. The downside of the flash device is the added complexity of managing the additional time required to program each page in the device. Erasure must occur on a sector basis or the entire device can be cleared in one operation.

Support for the 102-73161-25 design includes the capability to store both audio waveforms and commands.

2.2.5 Easy Synchronization

There should be a method of synchronizing multiple transmitters.

This is not to imply that the transmitters must be in contact with each other.

330 Implementation:
The transmitters are synchronized a day or two prior to the event through the command port.

The network path is deprecated in the V3 and later software as it was never used in the field.

2.2.6 Long Battery Life

This is to say we should be able to operate for the entire length of the fox hunt without having to replace batteries.

341 Implementation:
The case has room for a 6 cell AAA battery pack. This should allow for 12 to 24 hours of operation of the zNEO based transmitters.

3 Theory of Operation

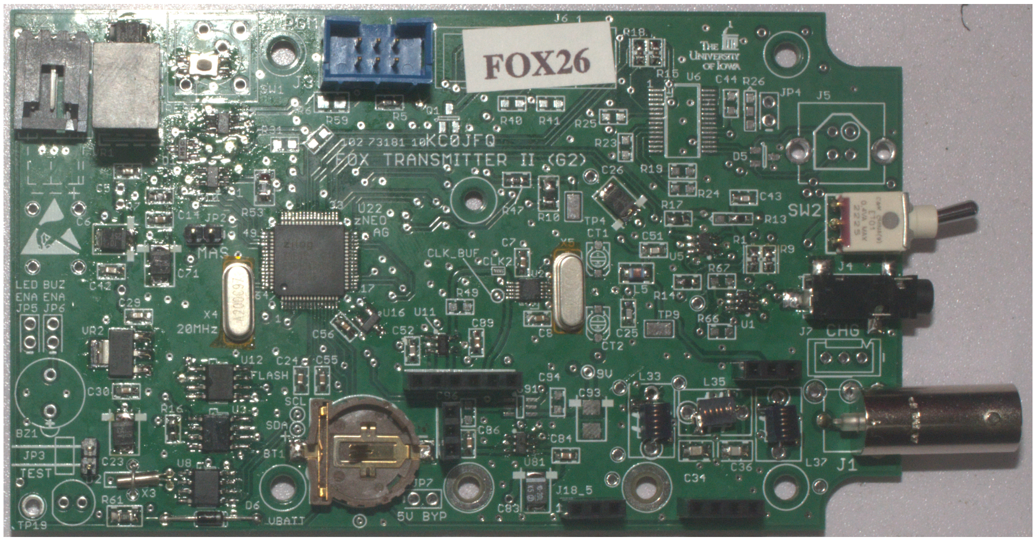


Figure 1: 3^{rd.} generation transmitter (102-73181-10)

Incorporating all the good features of the previous units. The 102-73161-25 and the 102-73181 boards are similar enought to allow this software to operate all of them.
The 102-73161-7 and 102-73161-12 boards are dissimilar enough to not be compatible.

Note that the 3.5mm jack, center right. This provides for a lower cost serial connection to load the operating sequences into the transmitter. The command jack on the 102-73181-5 artwork requires a haywire (on the backside of the board) to correctly receive command traffic. The 102-73181-10 reallocates the second serial port to control the RF modlue (DRA818/SA818 or handie talkie).

The USB UART pads are retained on the board, although it will not be normally be populated to reduce cost and improve useability. This is the configuration shown in the image, reducing the per-unit cost a bit.

This iteration adds a circuit to keep the backup battery charged when a battery pack is attached to the board.

The 102-73181-10 revision was necessitated by lack of availability of the 80 pin package that housed the zNEO processor. This revision moves to a 64 pin package but remains software compatible.

A few hardware improvements were added that do not change the function of the system. One change is moving the daughter board power control to its own pin. This change allows for reliable functioning of the DRA818/SA818 daughter board.

z

3.1 RF Overview

The motherboard RF section uses a programmable clock synthesizer to generate a carrier in the 2M band. This clock synthesizer makes use of a Digital Phase Locked Loop (DPLL) to generate a clock signal in the 2M band. The reference clock for clock synthesizer is generated using a simple 20MHz crystal (identical to the crystal used by the zNEO). Audio modulation is obtained by varying the load capacitors across crystal using the audio signal to drive a pair of varactors.

The modulated reference clock is multiplied using an oscillator internal to the clock synthesizer and then using the DPLL to slave the generated clock to the reference clock from the 20MHz crystal. The output of the clock synthesizer is a square wave that is sent to the RF daughter board. transmitter.

An amplifier may appear on the RF daughter board to increase the amplitude of the clock synthesizer output. The RF daughter board is free to implement an RF amplifier as it sees fit.

The output from the RF daughter board, routed back to the main board, is then filtered to remove harmonics above the operating frequency.

The RF daughter board may also be fitted with a VHF transceiver module. This arrangement eliminates the clock synthesizer altogether. The VHF transceiver module is commanded using the serial path that connected to the clock network on earlier revisions of the board.

Note that the transceiver module is available in UHF variants, allowing the system to be used in the 23cM band.

3.1.1 Motherboard Transmitter section

The motherboard transmitter section is built around the SI5351A clock synthesizer. The SI5351A has a built-in oscillator that is used to generate a reference clock that is, in turn, used to discipline an internal VCO. This internal VCO is configured to operate at the carrier frequency and fed, along with the reference clock, into a DPLL. The output of this internal DPLL keeps the internal VCO operating at the target frequency.

The SI5351A built-in oscillator uses an external crystal to generate a stable reference clock. The load capacitors on this crystal are varactors (voltage variable capacitors) that are, in turn, controlled by the audio modulation signal. The varying audio signal changes the load capacitance on the crystal to shift its operating frequency, thus (FM) modulating the carrier.

For low power applications, the amplifier stage is left unpopulated and the output of the SI5351 is delivered directly to the output filter. Output power using only the digital output of the SI5351 is about 10mW. This output level should be adequate for short range hunting.

3.1.2 VHF Transceiver Module

There are at least two VHF transceiver modules available (eBay or Amazon) that provide an almost complete walkie-talkie function, These are the DRA818 and SA818 modules. These modules have the entire RF section for transmitter as well as receiver. The RF circuitry on the main board need not be populated when making use of these Tx/Rx modules.

For a our fox transmitters the receive section is not used. The module may be pushed into a power-down state or switched off to conserve power between transmissions.

3.1.3 Output Filter

There is an output filter on the main board to reduce harmonics produced by using the simple square wave clocks from the SI5351A. The DRA818 and SA818 modules also produce rich harmonics when transmitting. The motherboard filter component size is fixed on the artwork. The chosen inductors should allow for a 6 meter filter to be configured with values noted in the table on the schematic.

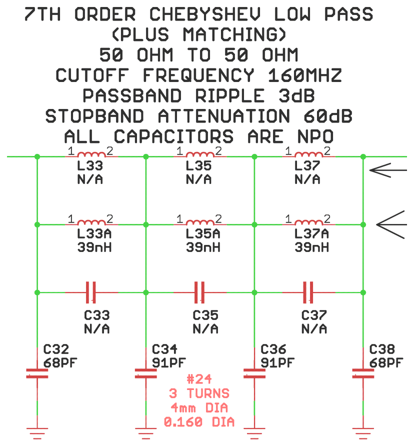


Figure 2: SI5351 Output Filter Schematic

Note that the board is provisioned with extra parts to allow the filter configuration and parts selection to change.

3.2 Frequency Selection

There are several frequency selection methods that will be encountered with the 102-73161 and 102-73181 hardware.

There is a limited frequency selection table in the fox transmitter operating software. These tables, for the SI5351 and ICS525, cover a very limited frequency range.

The software also supports a method of saving register values in the flash. Operating on frequencies outside those provided in the table is easily accomplished by storing register values in the FRAM.

The DRA818/SA818 modules are directly programmed with their operating frequency using the **FREQ** command argument.

3.2.1 SI5351

The SI5351 requires us to come up with a set of control patterns. This device has a bit less than 188 addressable registers.

Skyworks provides a clock configuration tool for the SI5351.

The plan is to generate the SI5351 setup patterns externally. A small subset of frequencies are held in the program flash and the register patterns may be set through the command interface.

This approach allows convenient operation for a limits frequency set while retaining the capability of setting the SI5e51 to any arbitrary frequency.

Keep in mind that there are no interlocks in the software to limit the frequency we select.

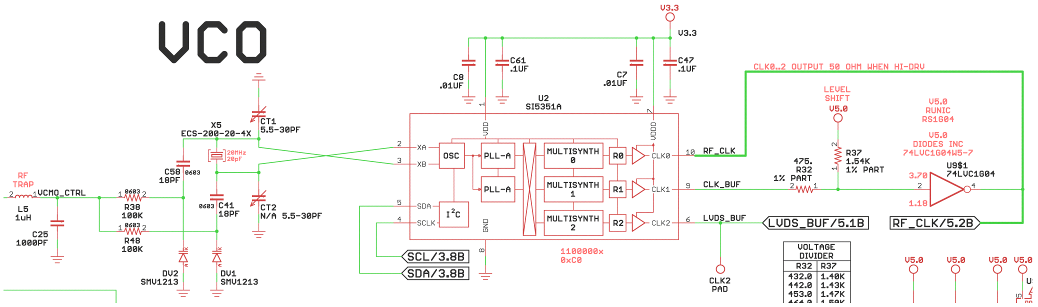


Figure 3: SI5351 Schematic

On the left you will find the FM modulation scheme introduced on the 102-73161-25 board. A pair of varactors diodes are used to provide the capacitive load for the reference clock crystal. The control voltage arrives at L5 on the left of the schematic fragment.

Trimmers are provisioned, although the SI5351 provides internal load capacitors that are programmable, so we don't expect to populate CT1 or CT2.

The SI5351 synthesizer, U2, provides three clock channels that can be independently configured. In our application, however, we will configure all three synthesizer blocks to produce the same frequency, that is to say the carrier.

Only one of the PLL synthesizers is required as we are operating all three channels at the same frequency.

The clock output that is in use is enabled using a command sent to the I^{2}C port on the SI5351.

CLK0 is a direct drive to the RF daughter board. To use this connection on the 102-73181-5 card, a haywire must be installed to form the connection.

When the clock output is configured for high drive (i.e. 8mA), the nominal output impedance is 50Ω. The configuration bits are in SI5351 register 17.

CLK1 is buffered and shifted to a 5V level before being routed to the daughter board.

CLK2 is buffered using an LVDS driver and the delivered to the daughter board.

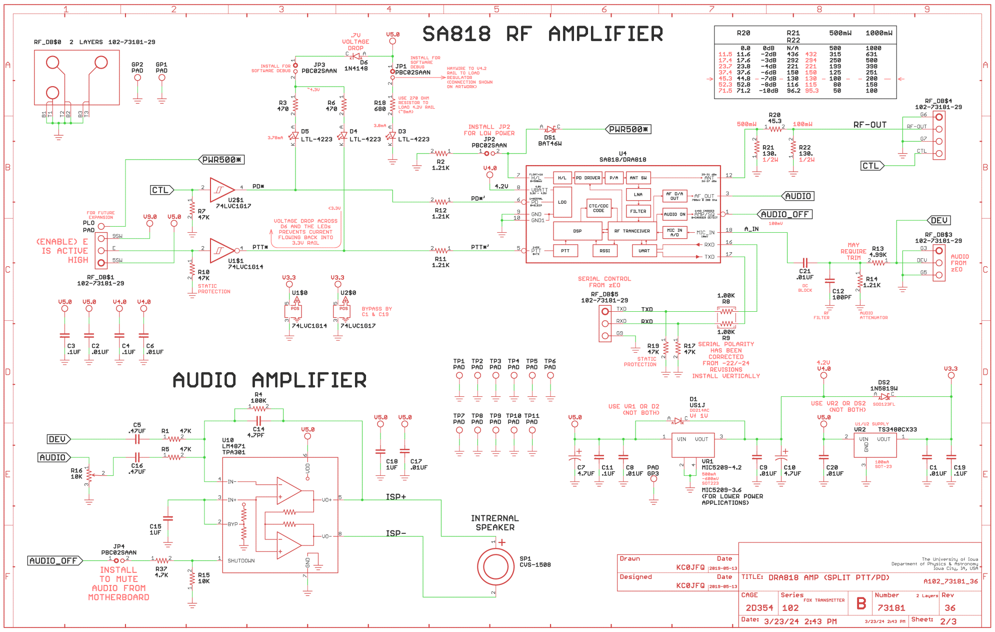
Take note that on the 102-73181-10 card, CLK0 and CLK1 share a pin on the daughter board connector. If U4/U9 are populated, a trace cut is required on the bottom side of the circuit board.

3.2.2 SA818/DRA818

The SA818/DRA818 modules differ in that they are self contained radio subsystems. Frequency selection is handled in the module, not by the clock synthesizer of the motherboard. The ICS307 does not even need to be populated when using this RF module.

The module is controlled using the UART channel that is routed to the time network. A switch is added to the 102-73181 board to allow the zNEO to listen to either the time network input or to the SA818/DRA818. The output channel connects to both the time network and to the SA818/DRA818 module. The time distribution function and SA818/DRA818 commanding peacefully coexist, time messages and SA818/DRA818 commands both appear at the SA818/DRA818 device and on the time network. The SA818/DRA818 ignores time messages and any transmitters connected to the time network ignore SA818/DRA818 traffic.

Selecting frequency on the SA818/DRA818 may be done directly, with the zNEO simply limiting selection to the 2M and 70cM amateur spectrum (the SA818/DRA818 may be had in either VHF or UHF models)



	R20		R21 R22		500mW	1000mW
	0.0	0dB	N/A		500	1000
11.5	11.6	-2dB	436	432	315	631
17.4	17.6	-3dB	292	294	250	500
23.7	23.8	-4dB	221	221	199	398
37.4	37.6	-6dB	150	150	125	251
45.3	44.8	-7dB	130	130	100	200
52.3	52.8	-8dB	116	115	80	158
71.5	71.2	-10dB	96.2	95.3	50	100

658

Figure 4: SA818 Daughter board Schematic

This is the unmarked schematic. A production unit would not have the audio amplifier and its associated parts installed. Neither would the indicator LEDs be installed.

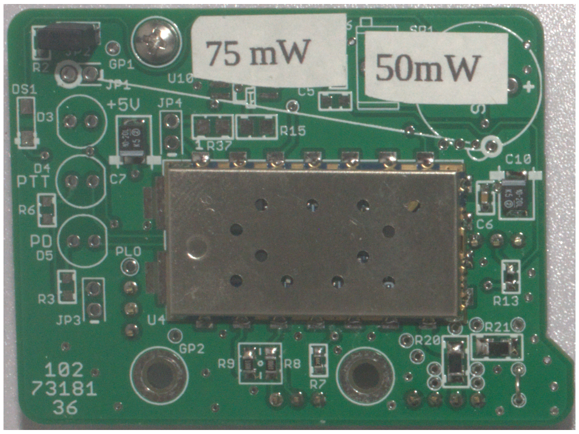


Figure 5: SA818 Daughter board

This is a functional SA818 board. This is the latest revision.

The power labels show the power delivered to the antenna after the signal is attenuated (R20, R21, R22 on the 102-73181-36 daughter board) and filtered (on the 102-73181-10 motherboard).

The two label values indicate power measured without the JP2 jumper in place and the power with the JP2 jumper in place.

The DRA818/SA818 daughter boards operate only with the 102-73181-10 circuit board. Earlier main boards do not provide power control that allows the DRA818/SA818 to operate reliably. The control software manages **PTT***, **CTL**, and power to the board independantly. The V3.27 and later software correctly supports power sequencing.

3.2.3 ICS525

The ICS525 is a simple clock generator aimed at producing additional clocks requirede in a multi-clock domain. We hijack it to produce several frequencies in the 2M band.

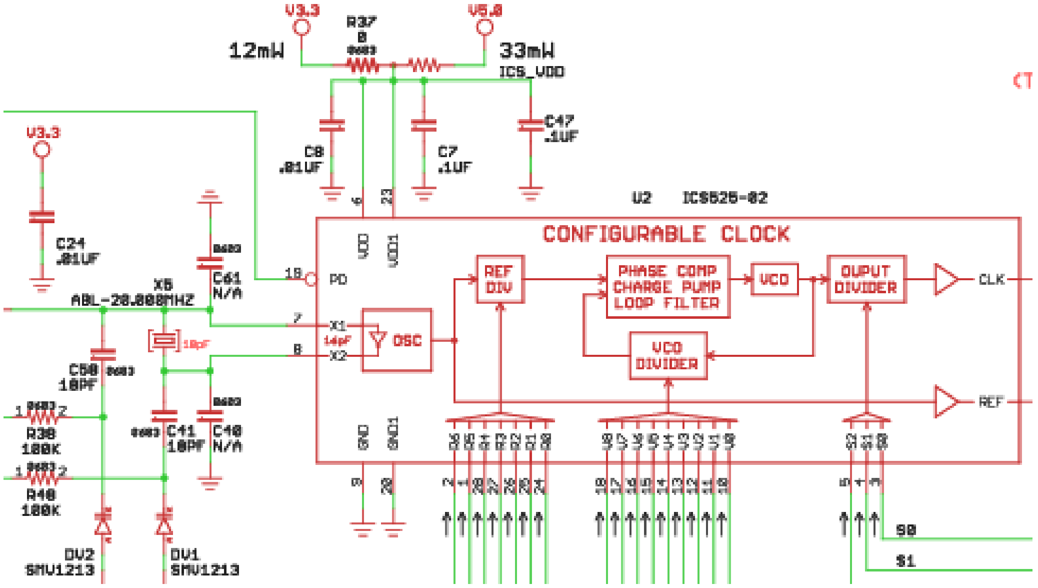


Figure 6: ICS525 Schematic

The modulation scheme was first tested on the 102-73161-25 revision. DV1 and DV2 provide the variable load on the crystal while C58 and C41 provide DC isolation.

The ICS525 requires 3 divider values to operate. A small selection of values are stored in an internal table to support the 102-73161-25 hardware. Each entry has a frequency value (floating point), and the three register values for the ICS525. The selection method is to find the closest value in the table. In practice, the table may be dumped to find the available frequencies (**D525 TABLE**).

Selecting the **ICS525** device with the **CONF** command also sets configurations for the port bits, analog channels available for use, and the analog reference used by the zNEO A/D subsystem.

The VCO configuration on the 102-73161-25 board is more-or-less identical to that shown in figure **??** on page 8.

The ICS525 module may not be present in the image loaded into the 102-73181 boards. This saves a small amount of ROM space.

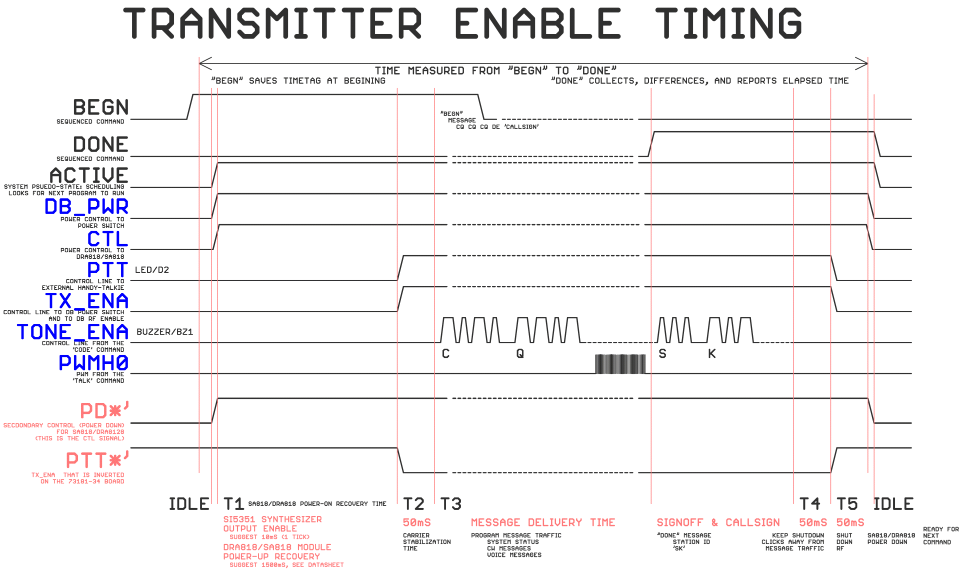
3.2.4 ICS307

Not currently supported as no boards were built using the ICS307 device...

3.2.5 Transmit Timing

A transmission cycle is initiated when a **BEGN** command is executed. This command turns on the RF system and waits for it to stabilize. Once operating steady-state and identification messages is transmitted.

Transmission is terminated after a **DONE** command is executed. This command causes a station ID to be send (i.e. the station callsign) before the RF system is shutdown.



TRANSMITTER ENABLE TIMING
TIME MEASURED FROM "BEGN" TO "DONE"
"BEGN" SAVES TIMETAG AT BEGINING
"DONE" COLLECTS, DIFFERENCES, AND REPORTS ELAPSED TIME
BEGN
SEQUENCED COMMAND
"BEGN" MESSAGE CQ CQ CQ DE 'CALLSIGN'
DONE
SEQUENCED COMMAND
ACTIVE
SYSTEM PSUEDO-STATE: SCHEDULING LOOKS FOR NEXT PROGRAM TO RUN
DB_PWR
POWER CONTROL TO POWER SWITCH
CTL
POWER CONTROL TO DRA818/SA818
PTT
LED/D2
CONTROL LINE TO EXTERNAL HANDY-TALKIE
TX_ENA
CONTROL LINE TO DB POWER SWITCH AND TO DB RF ENABLE
TONE_ENA
BUZZER/BZ1
CONTROL LINE FROM THE 'CODE' COMMAND
C
Q
S
K
PWMH0
PWM FROM THE 'TALK' COMMAND
PD*'
SECDONDARY CONTROL (POWER DOWN) FOR SA818/DRA8128 (THIS IS THE CTL SIGNAL)
PTT*'
TX_ENA THAT IS INVERTED ON THE 73181-34 BOARD
IDLE
T1
SA818/DRA818 POWER-ON RECOVERY TIME
SI5351 SYNTHESIZER OUTPUT ENABLE
SUGGEST 10mS (1 TICK)
DRA818/SA818 MODULE POWER-UP RECOVERY
SUGGEST 1500mS, SEE DATASHEET
T2
50mS
CARRIER STABILIZATION TIME
T3
MESSAGE DELIVERY TIME
PROGRAM MESSAGE TRAFFIC
SYSTEM STATUS
CW MESSAGES
VOICE MESSAGES
SIGNOFF & CALLSIGN
"DONE" MESSAGE STATION ID 'SK'
T4
50mS
KEEP SHUTDOWN CLICKS AWAY FROM MESSAGE TRAFFIC
T5
50mS
SHUT DOWN RF
SA818/DRA818 POWER DOWN
IDLE
READY FOR NEXT COMMAND

Figure 7: Transmit Timing

BEGN Command

This command may take some time to execute when using the DRA818/SA818 as it manages the timing to place the transmitter into an active state and send the *signon message* traffic. The simple RF amplifier boards so not require the extended wakeup time.

This *signon message* looks like:

```
CQ CQ CE de <call>
```

All message traffic starts this way so that an initial callsign is sent before any variable message traffic.

This is the timing chart for the DRA818/SA818 which has an extended T1 period to allow the transceiver module to come out of a sleep state. When using the SI5351 synthesizer, this T1 period is considerably shorter as the SI5351 requires on the order of a few milliseconds to start producing the carrier.

This command *intrudes* into the beginning of the T3 period as it sends out its signon traffic.

DONE Command

This command also takes some time to execute as it manages the timing to shutdown the transmitter at the end of message traffic. A *signoff message* is sent that looks like:

```
<call> de SK SK SK
```

The *signoff message* and station identification must be sent while the RF system is active. Once all the required traffic has been sent, the RF system is placed back into an idle (low power) state.

The station identification serves to keep the unit compliant with part-97 requirements.

ACTIVE System State

This trace indicates when the Fox Transmitter System is dealing with active message traffic. This is an active scheduling event. No other scheduling activities may occur at the same time (i.e. single threaded execution).

TX_ENA Motherboard Signal Net

This is the control signal (from the zNEO) that causes the transmitter to produce RF. When this signal is active, the 5-volt and 9-volt(battery voltage) supplies to the daughter board are enabled.

This signal enables the power switch devices U81 and U91.

CTL/PD*' Motherboard Signal Net/SA818 Signal Net

This is the control signal (from the zNEO) that has been added to deal with the power down feature of the DRA818/SA818 module. The logic level of this signal may be changed on the SA818 daughter board.

PTT*' SA818 Signal Net

This is the **TX_ENA** motherboard signal, which is positive true, with its logic sense switched to match the requirement of the SA818.

T1 SA818 module wakeup time.

This is the time needed for the SA818 module to exit the power down state in preparation for transmit. Note that this time is somewhat lengthy.

This timing element is set according to the RF subsystem.

T2 RF stabilization time.

This is time required for the transmitter to stabilize after being enabled (i.e. after PD active). This is the time period needed before the RF subsystem to produce clear audio as well as a guard time to allow the receiver to open its squelch circuit.

This timing element is set according to the RF subsystem.

T3 Message Transmission Time.

This is the message delivery period. The time spent in this state is controlled by the message traffic.

This timing is set by the outgoing message traffic.

T4 Shutdown Quiet Time.

This time slot follows the shutdown message traffic. This provides a guard at the end of transmission so as not to chop off the end of the shutdown message.

This timing element is set according to the RF subsystem.

T5 Shutdown time.

This is the time period following the shutdown of the RF subsystem. It provides an interlock time between messages.

This timing element is set according to the RF subsystem.

3.3 Voice

A voice capability was not initially planned for 102-73161 boards, Although this would have been very desirable, there didn't seem to be enough resources to be able to provide this capability using the zNEO.

Following some *deep thought* following a question from W0PPF an answer began to surface (*42*). With a bit of experimenting, the PWM controller in the zNEO provided the solution.

Using a large FLASH device (64Mb) we can store about 1000 seconds of audio that is digitized with 2KHz of bandwidth.

The zNEO is extremely SRAM limited, so the waveforms are not buffered in the zNEO but are processed a single sample at a time. The sample rate is controlled by the shift clock on the SPI bus, this clock being configured to operate at 32KHz resulting in a new sample every 250uS.

The audio file can be re-sampled into the required format using *sox*. The expected waveform is 8 bit unsigned. The sample rate may be set between 4,000 samples/second, and 16,000 samples/second. The audio bandwidth of the RF section is quite limited so the slow sample rate (4KHz) results in understandable speech. This slow sample rate also allows a reasonable amount of voice storage.

Increasing to 5KHz sample rate modestly improves readability without significantly decreasing storage capacity (in terms of time stored in the flash device).

A sample rate of 8KHz cuts the available time in the flash device while increasing bandwidth past what is allowed on the 2 meter band. This bandwidth will also be limited by the filter that sits between the PWM output and the modulation net.

Sample rates of 10KHz and 16KHz may also be processed from the FLASH memory. These sampling rates exceed the allowed bandwidth available through the RF section, so should not be used in a Fox Transmitter application.

These sample rates are implemented to allow the Fox Transmitter motherboard to be in other applicaitons where clear audio is required. At these sample rates, the volume of data loaded into the FLASH becomes much larger. This requires larger FLASH devices and increases load times.

3.3.1 Audio File System

The *Audio File System* in the zNEO system resides in the FLASH device. It only shares space with the *Configuration Command File System* for storing the directory.

Separate device erase commands are implemented for the FRAM and FLASH. These commands do not cross device boundaries.

The *Audio File System* directory consists of multiple 32 byte records in the *Configuration Command File System*. The *Audio Date* is stored in the FLASH device as binary data. Each file consisting of some multiple 32 bytes.

The data in the *Audio File System* is loaded using *Intel HEX* records. The address in each *Intel HEX* record is relative to the start of the FLASH device. As the FLASH device is larger then 64K bytes, a type-4 *Intel HEX* record is used to deal with the upper bits of the address.

The command decoder recognizes an *Intel HEX* record by its leading colon (:) character.

Example portions of an audio file:

```
:02 0000 04 0000 FA
:20 0000 00 52494646501000005741564566 6D742010000000001000100A00F0000A00F0000 4F
. . .
:20 3FE0 00 8A86757E787D8B898A8C807C7E717764708E749F8C7D916D777A76808D828C89 B6
:02 0000 04 0000 FA
:20 4000 00 7883767A8489888C837D7F7677766A82847D9B7E88876F8077788584848D7F82 A4
. . .
:20 F440 00 7C7D7D7C7E7C7D7F7E7F7E7D7D7D7C7D7C7D7D7C7D7E7C7C7E7C7C7D7C7E7D7D 0D
:00 0000 01 FF
esav TALK=V_W0JV 56448
```

This audio file fragment shows waveform image (i.e. the Intel HEX records) and a directory entry describing location of the audio file.

This example has a vocalization of the W0JV callsign. This audio fragment is RIFF/WAVE formatted, so the length and sample rate are in the RIFF/WAVE header.

Had the file been a naked waveform, more information would be required in the directory record. It would then have to appear as:

```
esav TALK=V_W0JV 56448 6112 4K
```

With the length and sample rate explicitly specified in the directory record.

Table 1: Intel HEX record

Column Group	**Contents**	**Column Description**
1	Length	Intel HEX record Key and record length
2	Address	Address (within 64KB page)
3	Type	record type
4	Data	core image data
5	Checksum	simple sum of all hexadecimal data

The above example was produced by the audio utility that is used to gather multiple audio clips together to be stored into the FRAM and FLASH devices.

The *Intel HEX* record decoder ignores white-space within the hes record. The audio utility inserts the white-space to delineate the columns indicated in Table .

Table 2: Intel HEX record Types

Type	**Record Contents**
0	ASCII encoded HEX record
1	END-of-FILE record
2	Address Offset
4	Extended Address

These are the *Intel HEX* record types recognized by the command decoder.

The zNEO deals with the *Audio File System* through the **TALKJ=** record in the FRAM file system. This TALK record holds the name, start address, byte count, and sample rate of the audio clip. When commanded to speak, the zNEO looks in the FRAM file system for a matching name and uses the start, length, and rate to read data from the FLASH memory and pass it into the PWM register that controls the PWM width.

The rate with which the data is processed is controlled by the SPI clock rate. The SPI clock is programmed to operate at eight time the sample rate. This causes the SPI controller to shift data at the correct rate. The data available status in the SPI controller regulates the sampling rate.

3.3.2 Audio File Utility

The *Audio File Utility* is used to gather a number of audio clips together and produce an Intel Hex File that can be downloaded into FRAM.

Input to the *Audio File Utility* is a flat file listing the audio clips that will be loaded into the fox. Output consists of two files, an *Intel Hex File* containing the waveform data, and a directory file that contains the directory records for the waveforms.

The raw input files are produced using *sox* to re-sample an audio clip, typically in the form of a .wav file, to a 4KHz sample rate, 8 bit unsigned samples. The resulting file from this processing is reformatted by the *Audio File Utility* for loading into the fox transmitter.

The *Intel Hex File* produced uses three record types; a *type-0* detail record that is 32 octets long, a *type-4* extended address record to provide the upper address bits, and a a *type-1* EOF record to indicate end of file.

3.4 TOY Clock

The TOY clock is used to keep track time. This allows the group of transmitters to setup prior to use and then activated (i.e. switch on) without the need to perform any synchronization at the event. Careful selection of the load capacitors on the DS1672 clock chip should allow the clock to run with minimal error.

3.4.1 The DS1672

The DS1672 is a simple TOY (Time-of-Year) clock. It is a 32KHz oscillator and divider. A 32 bit seconds counter is presented over the I^{2}C interface. This 32 bit register may be loaded by the zNEO to set time.

The DS1672 manages a backup battery, so an on-board battery is provisioned on the circuit board to keep the clock running when the system is not powered.

3.4.2 The DS1672 Charge Control Circuit

A rudimentary backup battery maintenance circuit is included starting on the 102-73181-5 board revision. This circuit supplies operating current to the DS1672 when the main battery is connected. Current draw from the main battery is minimal, on the order of about 25 micro-amps.

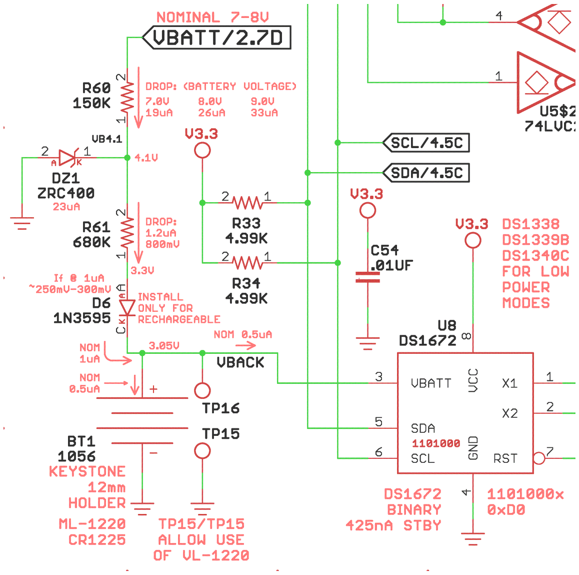


Figure 8: DS1672 Charge Control

The backup battery maintenance circuit, show above, provides operating current to the DS1672 and a few hundred nano-amps to the backup battery. When connected to the main battery, the coin cell should remain fully charged.

As the current is limited to around 1 micro-amp, a primary cell with a *reverse current* limit of 1uA may be used when a battery holder is installed in place of a permanently installed cell.

The main battery voltage starts out at a bit over 9.0V (i.e. when fresh batteries are first installed). R60 and DZ1 form a shunt regulator to provide a fixed voltage node to then supply a (more-or-less) fixed current to the battery and DS1672.

The voltage at the cathode of DZ1 remains fixed, so the current through R60 varies with the main battery voltage.

This fixed voltage at the DZ1 cathode allows the current through R61 to be independent of the main battery voltage. The resulting current is primarily dependent on the backup battery voltage, D6 and the resulting voltage drop across R61.

D6 isolates the backup battery from any loads that may appear on the **VBATT** net. The only current that comes out of the backup battery goes to the DS1672

The forward voltage drop across D6 was experimentally measured at approximately 250mV. This leaves the drop across R61 at roughly 800mV. Given the value of R61 at about 680KΩ, the current supplied to the coin cell and the DS1672 is limited to about 1.2uA. Roughly half of this is consumed by the DS1672, leaving less than 1uA of charge current available to charge the coin cell.

Although the circuit will supply standby current when the zNEO will not run, the transmitter should not be left with discharged batteries in order to avoid battery leaks that will cause damage to the circuit board.

3.5 Deviation Control

Deviation signal generation.

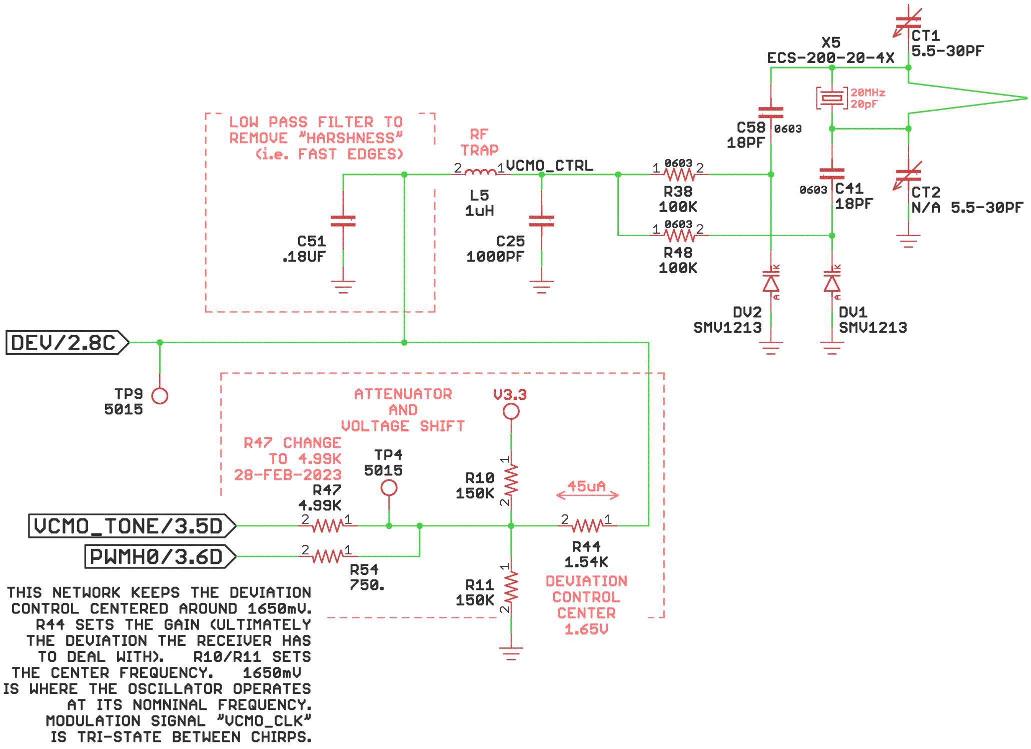


Figure 9: Deviation

The output from R38/R48 connects to the crystal load capacitors shown in figure **??** on page 8.

VCMO_TONE is the CW signal from the square wave generator (in the zNEO).

PWMH0 is the audio signal from the PWM channel in the zNEO.

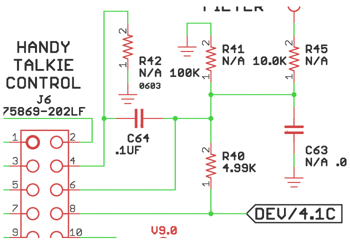


Figure 10: External COnnection

On the right, note that the DEV signal touches R40/R41. These two resistors may be used in conjunction with R44 to attenuate the signal that appears on the tuning capacitors on the SI5351 crystal.

The DEV (audio) net is also routed to the RF daughterboard.

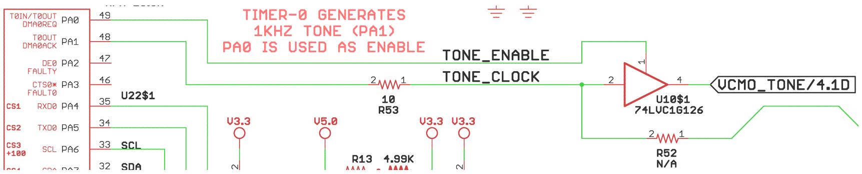


Figure 11: VCMO_TONE

The CW audio tone comes from a counter/timer block in the zNEO. Timer 0 is configured to generate a square ware with the period coming from the **TONE** command. This square wave is output on pin 48 and is buffered by U10. PA0 (on pin 49) is an output bit used to gate the output of PA1 (on pin 48). This reduces (slightly) the number of instructions needed to send code.

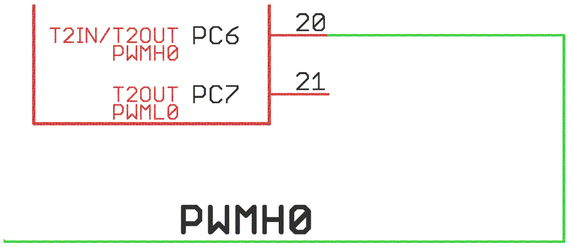


Figure 12: PWMH0

The audio signal is produced by the PWM controller in the zNEO. PC6 is configured as the output of the PWM block.

3.6 Configuration Order

Configuration commands are somewhat order sensitive and can be found by entering the **CONF** command with no arguments.

The *callsign* and *nickname* should be located at the begining of the **INI=** file. This establishes the identity of the unit for all commands that follow.

Following this device selection, the configuration bits are pre-set to appropriate values for the selected RF subsystem. Additional **CONF** commands may be issued to alter the RF configuration bit settings. The commands to do this are in the *SYNTH* group.

Any additional changes to the configuration bits com in the *RADIO*, *AUDIO* and *ANALOG* groups.

An external walkie talkie and the SA818/DRA818 modules are commanded using the secondary serial channel. You may find it necessary to alter some of the *RADIO* settings when using an external walkie talkie.

3.7 Power

The device is aimed at being able to run on a LR22 9V alkaline battery for a minimum of 8 hours with a 20% duty cycle. The circuit board in the target case allows for using a AAA pack to increase battery life to well in excess of 24 hours. It also seems that a set of six AAA cells are less expensive that a single LR22 battery, go figure.

The voltage regulator in the -7 and -12 artwork is a simple linear regulator so the efficiency is rather unremarkable. Operational life is somewhat less with the linear regulator.

The voltage regulator in later models and revisions is changed from a linear regulator to a switch-mode regulator in order to improve efficiency and, therefore, battery life. In addition, both the processor and the configurable clock are able to make use of a crystal to allow elimination of the MEMS oscillator. This change eliminates the most difficult to install surface mount parts and also reduces power consumption.

When using the higher power SA818/DRA818 RF board, keep in mind that the modules power requirement, when transmitting, is quite high; it is, after all, a 500mW/1000mW transmitter.
Do not expect a LR22 9V alkaline battery battery to supply enough current to turn on the SA818/DRA818 RF board, much less being able to transmit.

3.7.1 Battery

Recommended configuration is a six cell AAA configuration. Cells arranged as a 3x2 array will comfortably fit in the case listed in the build documents.

The 5V regulator is, however, capable of dealing with a higher input voltage than the nominal 9V provided by a six cell pack. The battery conversion coefficients may be changed using the **CONF** command.

> Using larger cells (i.e. AA cells) or switching to 8 or 10 cells can be employed to extend the operating life or to increase the operating power.
>
> The following peak voltages can be selected:

```
    CONF BMON 7.5V
    CONF BMON 10.0V
    CONF BMON 12.5V
    CONF BMON 13.5V
    CONF BMON 15.0V
    CONF BMON 17.0V
    CONF BMON 73161
```

The 102-73181 boards make use of an external voltage reference of 2.5V. The 102-73161 boards make use of an internal voltage reference. This, of course, affects the conversion coefficients.

The 102-73181-10 schematic has a table of resistor divider values for the above listed peak voltages. If the default value (**CONF BMON 10.0V**) is not to be used, R35 needs to be replaced with the targeted value.

Using an eight cell AA pack (located outside the case) would require this change and a **CONF BMON** command to change the coefficients used to calculate battery voltage.

CONF BMON 7.5V

This set of conversion coefficients is provided to allow a low voltage pack to be used. In particular, a 2-cell LiPo pack.

R35 is populated with a 10.0K Ohm resistor (R36 is always 4.99K Ohm).

Any of the higher voltage selections (for R35) may be used with a slight reduction in resolution of the reported voltage.

CONF BMON 10V

This is the default set of conversion coefficients that supports a 6 cell pack.

R35 is populated with a 15.0K Ohm resistor.

This provides maximum resolution when measuring a 6-cell pack.

CONF BMON 12.5V

This set of conversion coefficients are used to operate with an 8 cell pack.

R35 is populated with a 20.0K Ohm resistor.

When the fox transmitter has R35 changed to a new value, the coversion coefficient selection always matches the R35 value (**not** the actual pack attached to the transmitter).

CONF BMON 13.5V

This resistor selection is a better fit for operation with an 8 cell pack. This keeps the voltage at the input to the A/D below 2.5V when a fresh battery pack is installed.

R35 must be repopulated with a 21.5K Ohm resistor.

When the fox transmitter has R35 changed to a new value, the coversion coefficient selection must match the R35 value (**not** the actual pack attached to the transmitter).

CONF BMON 15V

This set of conversion coefficients are used to operate with a 10 cell pack.

R35 is populated with a 24.9K Ohm resistor.

As above, when the fox transmitter has R35 changed to a new value, the coversion coefficient selection follows R35. transmitter).

CONF BMON 17.0V

This resistor selection is a better fit for operation with an 8 cell pack. This keeps the voltage at the input to the A/D below 2.5V when a fresh battery pack is installed. The voltage at the A/D input is a bit high for the **CONF BMON 15V** values when using a 10-cell pack.

R35 is populated with a 28.7K Ohm resistor.

When the fox transmitter has R35 changed to a new value, the coversion coefficient selection must match the R35 value (**not** the actual pack attached to the transmitter).

CONF BMON 73161

This selection is used with the 102-73161 boards. Although the resistor values shown in the schematic are the same as the 102-73181 boards, the zNEO internal voltage reference is used.

3.7.2 Fuse

The 210-73181-10 board is provisioned with an 800mA fuse. This is packaged in a surface mount 0603 package.

The fuse is provided as part of the reverse polarity protection circuit. It should not ever be overloaded by the electronics. Replacement requires removal of a **small** soldered part on the bottom of the circuit board near the power switch. Investigate and correct any problem on the circuit board if the fuse ever requires service.

If the battery is connected backwards, D4 becomes forward biased which limits the reverse voltage to a diode drop (about 700mV) as the protection fuse, F1, is overloaded and pops. The current sense resistor R55 is temporarily stressed, but the fuse is expected to blow in a few hundred milliseconds.

3.7.3 Power Switching

Starting with the 102-73161-25 board revision, a power switch isolates the 5V and 9V rail to the daughter board. On board revisions 102-73181-5 and earlier, the switch is controlled by the **TX_ENA** net on port pin PH3.

The 102-73181-10 board seperates the power control function from the **TX_ENA** net, moving it to **DB_PWR** net on port pin PD7. A configuration resistor, R68, allows the power switch function to be selected to use **TX_ENA** or **DB_PWR**. For proper operation of the 102-73181-36 daughter board (i.e. the DRA818 VHF tranceiver module), the power switch must be independant of the **TX_ENA** net as the **TX_ENA** net is used on that daughter board to control the PTT function. Asswerting the PTT pin on the DRA818 as power is applied prevents the module from functioning correctly.

The simple RF amplifier modules, such as the 102-73181-35 or 102-73161-28, are not affected by having power and RF applied at the same time. They will tolerate R68 in either configuration. Do take note, however, that the power control is best left on the **DB_PWR** net as this is universally compatible with all RF modules.

The schematic for the 102-73181-10 board indicates that R68 would be installed to control U81/U91 from the **TX_ENA** net. Software development after the 102-73181-10 boards were fabricated indicates that R68 needs to be installed in the 2-3 position.

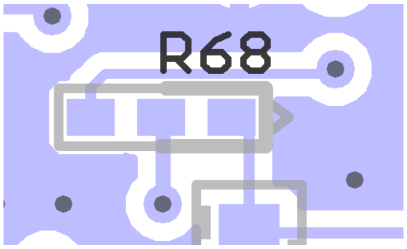


Figure 13: R68

The DRA818/SA818 modules will not have sufficient startup timer if R68 is installed to the right (the 1-2 position under the reference designator). If you encounter difficulty with the DRA818/SA818 modules, verify that R68 is installed to the left (the 2-3 position).

3.8 Host Interface

A host system is used to load the operating schedule and audio files. The board may be configured with a simple FTDI USB UART or with a 3.5mm stereo jack to connect to an FTDI **TTL-232R-3V3-AJ** serial cable. The USB connector and the serial connector are both oriented vertically on the 102-73181-5 to allow the battery access panel to be oriented toward the end where the antenna connector is located. This provides access to the USB port or 3.5mm jack when the battery compartment door is removed. The battery door does not allow to access a 6-cell AAA pack, so flipping things around should give convenient access to the serial port when preparing a 102-73181-5 unit for a hunt.

Figure 14: TTL-232R-3V3-AJ

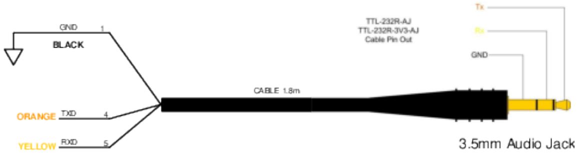


Figure 15: TTL-232R-3V3-AJ Pinout

The 102-73181-10 units move the 3.5mm serial port to the position vacated by the time network connector. This allows the 102-73181-10 board to mount to the shallow side of the case which is a bit more convenient.

The 102-73181-10 revision reassigns the handie-talkie serial port. It is now shared with the daughter board and uses the R1/R9 network to keep transmit and receive data on the correct pins. This isolates the handie-talkie from the zNEO to avoid static damage to the expensive 65 pin package.

3.9 External Radio

14-pin header: **J6**.

In addition to the 102-73181-36 RF daughter board, hardware is provided for controlling a hand held handie-talkie.

Table 3: J6 pinout and Function Table

14 pin	**Signal Name**	**Signal Description**
1	RXD0	Serial Traffic **from** external radio
2	TXD0	Serial Traffic **to** external radio
3	GND	circuit ground
4	VCMO_filtered	filtered and DC isolated VCMO_TONE
5	GND	circuit ground
6	VCMO_atten	filtered VCMO_TONE
7	GND	circuit ground
8	VCMO_TONE	PWM for voice operation simple square wave for CW
9	GND	circuit ground
0	PTT	Push-To-Talk negative true JP5 to enable pull-up
11	VBATT	switched battery before I-sense
12	V9.0	switched battery after I-sense
13	SWITCH	connected to zNEO PB4 analog input
14	PHOTO_CELL	connected to zNEO PB5 analog input

Note that the pin assignments are a super-set of the previous connectors on the 102-73161 boards. It would be possible to install a smaller connector on the board to accommodate an existing configuration, eliminating the need to rebuild external cables.

There weren't any board built and configured for external radio operation, so this is probably a moot point.

3.10 MASTER Jumper

The **MAS**ter jumper presents an interesting dilemma to the software. The 64 pin package used on the 73181-10 revision is one pin shy of matching up with the 73181-5 and earlier revisions. This problem pin is used to inspect the state of the **MAS**ter jumper.

The software must deal with this problem. The 64 pin package is missing the PF6 port bit and the hardware doesn't attempt to pretend that the PF6 exists internally. To remedy this, the software needs a method to detect which processor chip is present on the circuit board.

Fortunately, the ZiLOG engineers buried a device part number in the silicon that can be accessed by the software. Embedded in the part number is the package type, allowing the software to easily determine which of the revisions the software is running on.

The software copies the hardware part number from a reserved area of memory to SRAM and prints the resulting string as the system starts following a power-on or reset. The code that deals with the **MAS**ter jumper can inspect the saved part number and get jumper status from the appropriate port bit.

3.11 Processor

The zNEO processor is mechanically overkill in the 80 pin pin packages, but the 64 pin package is just about right (17 unused pins) and it is readily available.

As of 2023, the 80 pin package is like hens teeth. This drove the 102-73181-10 update to make use of the 64 pin package that is more available.

The only difference between the 64-pin and the 80-pin package that affects the design is the pin used to detect the **MASTER** jumper. The 102-73181-5 uses the PF6 bit and the 102-73181-10 uses the PF7 bit. The zNEO GPIO ports provide for a configurable pull-up that is used by the software to make the **MASTER** detect insensitive to the use of PF6 or PF7.

The zNEO processor is a 16 bit derivative of the ZiLOG Z8 with an architecture that is better tailored for use with compilers. The chip is available with 128KB flash and 4KB SRAM. Although smaller memory footprint devices are available, the current software will no longer fit the smaller footprint. Also, the cost difference is so small as to make it not worthwhile to even consider the smaller devices as they are only a dollar or two different in price.

The zNEO has a generous complement of peripheral devices that provide adequate resources for implementing the control system for the FOX Transmitter.

3.11.1 Software Toolchain Overview

ZiLOG provides a software development tool-set with a c compiler that is used to generate the applications software.

The compiler and linker are both able to run in a Linux environment using *WINE*. To program the device the the *ZDS II* IDE is required. Running under *WINE* limits programming to the *Ethernet Smart Cable*, but is does work under *WINE*.

3.11.2 zNEO Programming

A standard 6-pin header on the board allows the zNEO firmware to be updated.
There is no provision for programming through the USB port (that is not typically populated).

When using the ZiLog tools under *WINE*, the *Ethernet Smart Cable* is preferred to avoid problems configuring a *USB Smart Cable* or *Serial Smart Cable*.

The tools can, of course, bu used with Windows. This opens up the ability to make use of the *USB Smart Cable*.

The c-compiler supplied by ZiLOG has hardware-specific accommodations for the small SRAM footprint of the zNEO processor. In particular, **all** data that is static in nature (i.e. will never be written to), such as text strings, must be declared *ROM resident*. A declaration qualifier, *rom*, is used to keep data items and structures resident in the ROM. Text strings are declared with an **R** precedent:

```
            sprintf(R"format string", ...);
```

Inspecting the linker map should show no **NEAR_TEXT** allocations in user-written code sections. The only occurrences should be in library code. An example of the only occurrence of **NEAR_TEXT** in the V3 software is as follows:

```
Module: common\udtof.c (Library: fpsd.lib) Version: 1:0 08/07/2017 15:52:55
    Name                                          Base         Top      Size
    --------------------------------------- ----------- ----------- ---------
    Segment: CODE                             C:0167CC    C:016A61      296h
    Segment: NEAR_BSS                         R:FFB59C    R:FFB5A7        ch
    Segment: NEAR_DATA (was NEAR_TEXT)        R:FFB7F1    R:FFB800       10h
```

Careful attention to this has kept the available stack in the V3 software to almost 2K bytes (or half of the available SRAM).

3.12 FRAM and FLASH

The 102-73181 boards have two serial memory devices, both in 8-pin SOIC packages. The 102-73181-5 and later boards will (barely) accommodate wide packages.

The 102-73161-25 board has a single location for a serial memory device. This would imply that both audio and commands live together in one large ($$$) device.

3.12.1 Device Detection

The software has a device table that covers a reasonable number of FRAM and FLASH devices. The detection scan assumes that each of the two devices will respond to a JEDEC-ID request and return a three byte ID field.

The FRAM position (the IC labeled U3) will default to a 64Kb device if no JEDEC ID is found or no device is detected. This implies that the minimum size FRAM is, therefore, 64Kb.

3.12.2 FRAM

The software assumes the minimum size of an FRAM device (in location U3) is 64Kbits. Small devices that do not report a JEDEC ID will default to 64Kbit parameters.

The software rather assumes that an FRAM type device will be present in location U3 which holds the operating sequences. The use of an FRAM device allows for quick changes without having to erase and reload the entire device.

3.12.3 FLASH

No assumptions about the FLASH are made. All Flash devices must report JEDEC ID in order to be detected and have the appropriate access methods enabled.

The FLASH device (in location U12) holds audio data and would normally be loaded using the Intel HEX file interface. For this to work efficiently (and at a reasonable speed) a page-write device must be used here.

Buffer constraints in the zNEO limit the maximum Intel HEX record to 32 data bytes. This will result in a data record of 80 bytes when using the audio utility tools. A larger data payload will overflow the input buffer in the zNEO.

The audio file system is loaded all at once following a device erase.

3.13 Software Revisions

Updates and changes to the Operating Software

3.13.1 V3.62

Corrected a series of bugs caused by adding 32 bit support to the flash handler. Seems stable dealing with 256Mbit (and larger) flash devices. The address field increases to 32 bits (affecting read and write commands.

I also ran into long erase times while debugging the 32 bit support for the **Macronix MX25L256**. There isn't any special handling of a flash erase operation, you enter the **HERA ALL** command, it send the chip erase, and we're ready for the next command. The **MX25L256** on the other hand, is loafing along performing the chip erase and it won't talk to us while it's erasing. The chip looks like it's royally *borked* until the erase has completed.

All we do to deal with this is let the user know it's going to take a while. The flash commands test the ready bit in the status register and abort with a **BUSY** message to indicate the flash device is off doing something else.

3.13.2 V3.59

Well, we finally discovered that big FLASH devices require a 32 bit address. Add 32 bit address support to the flash routines.

We also encounter a l*oooooo*ng erase time where the flash device will not respond to anything other than a RDSR request to read the status register (and return a busy indicator).
We now detect a busy flash and report it afgter aborting the command request.

A late addition to *intel_hex.c* is a console break test into the dump loop. Bang on the enter key to break out of the loop (*which can go on forever with a large audio set*).

3.13.3 V3.57

Add a dummy modules for the ICS525.

> We don't need this block of code with the 102-73181 boards, so make a dummy module so it will link correctly...

3.13.4 V3.56

Remove MODW command and implement the MODC command.

The schedule state variable now has three operating states:

Idle

Schedule is loaded and ready to run, but is not currently scheduling.

Active

Schedule is loaded and is currently scheduling.

Clear

Schedule remains loaded and ready to run, but has been halted/stopped by the **MODC** command.

And a fault state where the state variable value is invalid.

Updated the CHRP command parameter handling.
CHRP <audio tone> <period> <duration> <repeat count>

3.13.5 V3.54

Rework the way the TEST and MAS jumpers are handled.

Always run INI= script (keeping callsign, nickname, and radio configuration commands in one place).

Both jumpers allow the system to recover from a bad command load. It suppressses all FRAM commands.
A minor revision was applied to the cmd_message.c module marked as V1.04 that adds a second delta in the **BEGN-DONE** calculation. When the **T1** delay exceeds 100 mS, 1 secondary delta time is displayed by the **DONE** command (this shows only the carrier-on time).

3.13.6 V3.53

Changes to BEGN, DONE, and WAIT to allowing the transmitrer for things other than fox hunting.

3.13.7 V3.52

Add sample rates to audio driver:
10K s/s and 16K s/s.

Useful in some non-FOX applications.

Add H56K command to speed up hex downloads...

3.13.8 V3.51

Rework of the ICS525 handler to correctly take register patterns from the frequency table in FRAM.

3.13.9 V3.50

Update to implement the **CHRP** command.

Emulate a wildlife tracker.

3.13.10 V3.31

Update that get waveform in FRAM sorted out.

This should make the 102-73161-25 units fully functional with the version3 software (at least, I hope it does).

3.13.11 V3.28

Fix ICS525 table dump.
Successfully tested on 102-73161-25 hardware.

> Also updated in this build, in the *V3.05 flash_local.c* module (Mar 5 2024 16:15:39), are a couple of updates to the FLASH tables (*device_table.c*) adding two FRAM devices. This update does not affect any current boards.

3.13.12 V3.30

Cleanup things...

Process CONF flags a bit more cleanly/completely. not present.

The **STAT** command has some changes to make it sensitive to analog channel enable flags.

3.13.13 V3.27

Changes in audio processing. We now detect and process RIFF/WAVE file headers to determine sample rate and sample count. Same subset of sample rates are implemented.

Old style entries still work for non RIFF/WAVE sample sets when the RIFF/WAVE file headers are not present.

3.13.14 V3.26

Change DRA818/SA818 startup time to 2000mS to accommodate slow turn-on of some variants.

3.13.15 V3.24

First test on the 102-73181-10 hardware.

> Update FLASH tables.
> Update GPIO_MASTER.

3.13.16 V3.23

The *cmd_stat* routine was clobbering the FRAM/FLASH select field when it scans the external memory devices as part of status reporting. A **STAT** command would cause an executing sequence to stop (fatal flaw!).

Add a save/restore entry point to *flash_local* to save all the location pointers for later restoration. Now *cmd_stat* calls the save and restore around the external memory device access.

3.13.17 V3.21

Release Date: Late Dec 2023.

Incorporating support for the 102-73161-25 boards. This artwork revision is similar enough to the 102-73181 boards that the software can support 102-73161-25, 102-73181-0, 102-73181-5 and the 102-73181-10 boards.

The 102-73181-0 boards do not exist *in the wild* due to the ICS307 being obsolete. As of this revision the ICS307 does not have any management code present.

3.13.18 V3.17

Release Date Dec 2023.

Fixed a bug in the CW generator. The BUG added 1 chip to everything, so code sounded funny and slow.

3.13.19 V3.16

Release Date Dec 2023.

Condense the battery reporting into the BATC and BATV commands.

3.13.20 V3.15

Release Date Dec 2023.

Implement the ONCE command.

3.13.21 V3.14

Release Date Dec 2023.

Updates to SI5351 support utility: *si5351_calc.c.* This utility updated to deal with an arbitrary Si5351 crystal, adds a frequency offset switch, and incorporates a new *MultiSynth* parameter calculation.

The **STAT** command is updated to display more SI5351 information when selected by the **CONF** command.

The **CONF** command updated to add SI5351 clock selection and clock drive keywords.

3.13.22 V3.12

Release Date Nov 2023.

DRA818/SA818 transitioned to be driven by the cmd_message.c module and not by the command.c module.

3.13.23 V3.10

Release Date 12 Nov 2023.

Merging in the SI5351 management code.

Formalize the T1, T2, T4, T5 timers that control transmit timing.

3.13.24 V3.03

Release Date 22 Feb 2023.

Continuing debugging.

Show transmitter type in **STAT** command.
Flash/FRAM handling now working.
Voice output now working.
Updates to *FOX_ICARC.tex*.

3.13.25 V3.02

Release Date 10 Feb 2023.

Major rework of the command decoder. The command dispatch table now has the address of the command handler and most of the command processing has been moved out of the *command.c* module into individual command processors. This methodology reduces the compiler overhead, significantly reducing the time required to recompile the software application.

The FRAM/FLASH system has been reworked to deal with both FRAM and FLASH devices and to split the file systems into commands and audio to take advantage of dual storage devices.

3.13.26 V2.00/2.01

Release Date 10 Jun 2020.

This is a development release that adds support for the DRA818/SA818 walkie talkie modules.

3.14 Hardware Revisions

Artwork and component selection.

3.14.1 102-73181-10

The zNEO is impossible to get in the 80-pin package. This reworks the circuit board to fit the 64 pin package.

This affects the **MASTER** signal. Software accommodates either the 73181-5 or the 73181-10 hardware.

Repurposed the network time port. It wasn't being used in the field and the SA818 module needs it's serial port. The 3.5mm jack is now connected to the serial command port so the case need not be opened to update the time prior to an event.

Added a small 800mA fuse to deal with a reversed battery connection.

The SI5351 can directly drive the RF daughter board with one of its clock pins.

Changed the 3.3V regulator to a higher power device (also less cost).

Add charge circuit to keep lithium coin cell topped up.

3.14.2 102-73181-5

Switch to yet another clock generator, the SI5351.

Add charge circuit to keep lithium coin cell topped up.

This revision will **not** work with the DRA818/SA818 module.

3.14.3 102-73181-0 w/DRA818

Switch to ICS307 clock generator, which is end-of-life (of course).

Add the 2nd. SPI memory device (FLASH) to allow for large audio files at lower cost.

Only one or two of these were built.

This revision will **not** work with the DRA818/SA818 module.

3.14.4 102-73161-25 w/DRA818

THis is the original series of boards that used the ICS525 clock synthesizer.

3.15 RF Amplifier Revisions

Artwork and component selection.

Current RF amplifiers in use:

1. **102-73161-22** Amp Bypass

   (See section 3.15.4 on page 40).

   Amplifier bypass. Synthesizer output direct to output filter.

2. **102-73161-23** SOT-89 MMIC

   (See section 3.15.5 on page 40).

   Class C amplifier using ADL5536 or similar. Part substitution required for DC block.

3. **102-73161-24** 74LVC04 gate

   (See section 3.15.6 on page 41).

   Class D amplifier using two 74LVC04 CMOS gates. All gates powered from 5V rail.

4. **102-73161-28** SOT-89 MMIC w/DC Block

   (See section 3.15.8 on page 42).

   Class C amplifier using ADL5536 or similar. DC block added to input and output pins of ADL5536. Input matching network. Input attenuatio network. Output matching network.

5. **102-73161-29** LVDS 74LVC04 gate

   (See section 3.15.10 on page 44).

   Class D amplifier using two 74LVC04 CMOS gates. Input is LVDS from SI5351.

6. **102-73181-28** SOT-89 MMIC w/DC Block, *CHIRP*

   (See section 3.15.9 on page 43).

   Same class C amplifier as 102-73161-28.

7. **102-73181-36** DRAQ818/SA818 RF MOdule

   (See section 3.15.14 on page 46).

   RF Module

3.15.1 102-73161-12S

MMIC Amplifier Adapter (not implemented).

This is a small patch board that may be installed in place of Q2 and C57. This replaces the MAX2602 RF transistor with a SMA3101 MMIC Amplifier. C57 moves the the patch board to provide adequate room to mount the patch board.

3.15.2 102-73161-12M

Bipolar Junction Transistor Adapter (not implemented).

This is a small patch board, identical to the -S board in size, that may be installed in place of Q2 and C57. This replaces the MAX2602 RF transistor with a BJT device in an SOT23 package.

3.15.3 102-73161-21 MCPH6 MMIC 50Ω amplifier

MMIC Power Amplifier Daughter board (not implemented).

This is a planned power amplifier that makes use of a SMA3101 or SMA3103 MMIC amplifier in an MCPH6 package.

3.15.4 102-73161-22 Amplifier Bypass

Low Power Daughter board.

This daughter board has an impedance matching network and an attenuator. The output of the ICS525 is routed through to the output filter. Output power is determined by the ICS525 supply voltage. Operating from the 3.3V rail yields about 12mW. Operating from the 5.0V rail yields about 33mW.

3.15.5 102-73161-23 SOT89 MMIC 50Ω amplifier

MMIC Power Amplifier Daughter board.

This power amplifier board makes use of an Analog Devices ADL5536 or ADL5544 MMIC amplifier in an SOT89 package.

A DC blocking cap is required at the input to the ADL5536/ADL5544 as this was missed in the artwork.

The ADL5536 is expected to produce close to 100mW.

The ADL5544 is expected to produce close to 50mW.

3.15.6 102-73161-24 60mW Class-D

74LVC Power Amplifier Daughter board.

This power amplifier board uses a simple CMOS gate to provide current gain and slightly increase the output power.

One 74LVC1G04W5-7 device buffers the RF Clock from the motherboard driving two 74LVC1G04W5-7 output buffers. These gates operate from the 5V rail and provide enough current to drive 60mW into a 50Ω load.

This particular gate was selected as being the fastest 74LVC gate currently available. The device is operating at the edge of its capability in the Amateur 2M band.

3.15.7 102-73161-27 90mW Class-D

74LVC Power Amplifier Daughter board (not implemented).

This power amplifier board uses a simple CMOS gate to provide current gain and slightly increase the output power.

One 74LVC1G04W5-7 device buffers the RF Clock from the motherboard driving three 74LVC1G04W5-7 output buffers. These gates operate from the 5V rail and provide enough current to drive 90mW into a 50Ω load.

3.15.8 102-73161-28 SOT89 MMIC 50Ω amplifier (50mW)

MMIC Power Amplifier Daughter board.

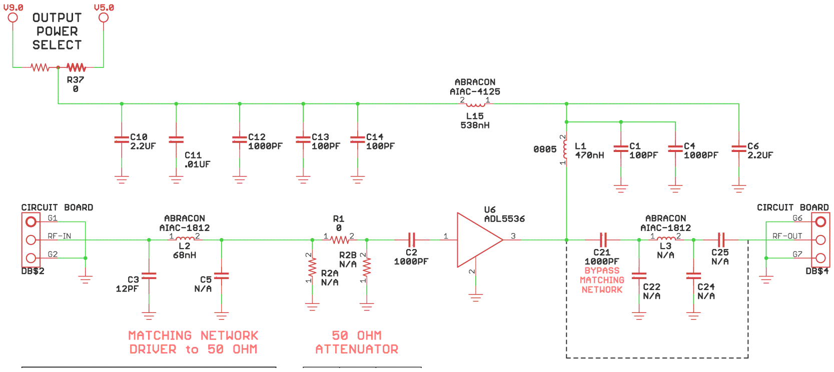


Figure 16: MMIC Amplifier Schematic

This power amplifier board makes use of an Analog Devices ADL5536 or ADL5544 MMIC amplifier in an SOT89 package. Several other MMIC devices may be used that have matching pinouts. The SI5351 provides adequate drive to produce the 50mW output level (use the CONF CLK0 to select the unbuffered clock).

C3, C5, and L2 form the input matching network that may be used to match the incoming signal to the 50Ω input of U6.

R1, R2A and R2B form an attenuator to reduce the amplitude of the incoming signal, if required. Our schematic shows R1 a 0Ω as we are not using this feature.

A missing DC blocking cap (C2) has been added to the artwork, superceding the 102-73161-23 artwork. Otherwise identical to the earlier board.

The output filter is not needed as the nominal output impedance of the amplifier is 50Ω, so C21 is installed across one pad of C21 and one pad of C25.

The ADL5536 is expected to produce close to 100mW.

The ADL5544 is expected to produce close to 50mW.

3.15.9 102-73181-28 SOT89 MMIC *CHIRP* amp

This is a rework of the 102-73161-28 amplifier specifically for interrupted carrier operation.

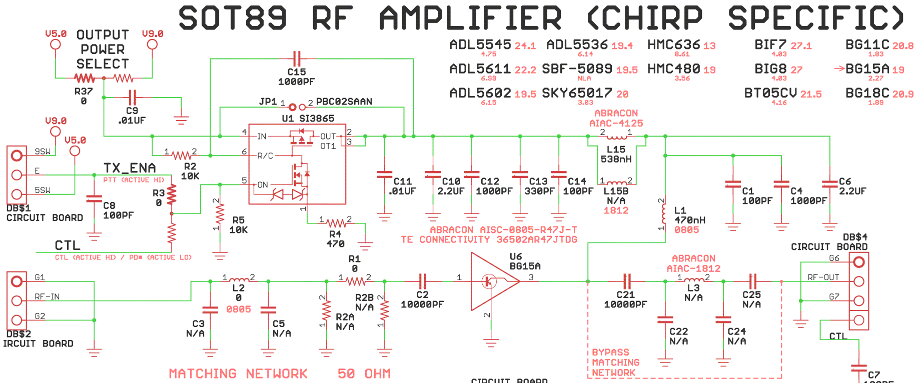


Figure 17: MMIC Amplifier Schematic

This rework adds a power switch (U1) to allow control of the power delivered to the MMIC amplifier. This emulates the operation of the DRA818 daughterboard where the PTT (push-to-talk) control is seperate from the PD (power down) control. The remainder of the circuit is essentially the same as the 102-73161-28 design.

The power jumper may be used on the motherboard (i.e. jumper J7) to move power control exclusively to the daughterboard. This isn't necessary; the jumper (J7) may be left off the motherboard to allow convenient interchange of different RF modules.

The default position of R3 is connecting the **TX_ENA** net to the control pin (U1-5) on the power switch.

A soft-start network, consisting of R2, R4, and C15, can have values adjusted to affect to dV/dT of the circuit.

3.15.10 102-73161-29 LVDS 60mW Class-D

74LVC Power Amplifier Daughter board.

This power amplifier board uses a simple CMOS gate to provide current gain and slightly increase the output power. over using a 102-73161-22 bypass board. The SI5351 on the 102-73181-5/102-73181-10 boards provide good drive to the LVDS driver on the motherboard (use the CONF CLK2 to select the LVDS clock).

One FIN1002 device buffers the differential clock from the motherboard driving a single 74LVC1G04W5-7 that shifts the 3.3V logic level to a 5.0V logic level. The 5V level then drives two 74LVC1G04W5-7 output buffers. These gates operate from the 5V rail and provide enough current to drive about 20mW into a 50Ω load.

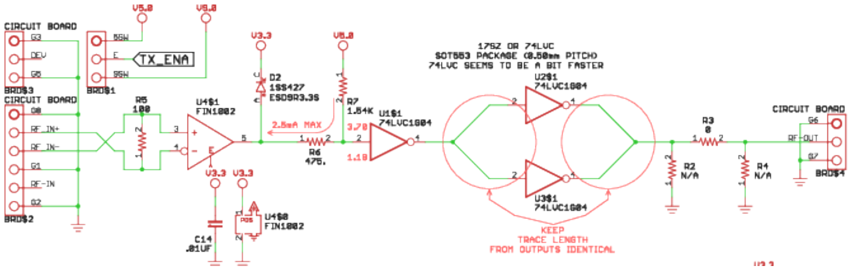


Figure 18: LVDS Amplifier Schematic

U1, U2, and U3 are all powered from the 5V rail.

The 5V rail is switched on the motherboard such that the board is only powered when transmitting.

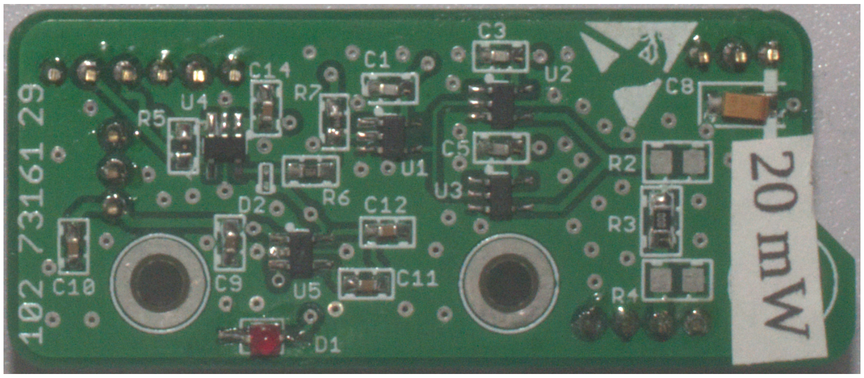


Figure 19: LVDS Amplifier

The output pin of U1 is located equidistant between the input pins of U2 and U3 to keep the trace lengths equal.
The output traces from U2 and U3 are also equidistant from the junction near R2.

R2, R3, and R4 form a PI network that may be used to attenuate the signal. These parts may also be used to match the output of the drivers to the filter on the motherboard.

3.15.11 102-73181-22 SA818 1W RF transceiver

SA818 or DRA818 RF module (deficient design).

This daughter board provisions an SA818 or DRA818 RF *walkie-talkie* module. These modules may be obtained in both VHF and UHF variants.

The SA818/DRA818 modules are self-contained RF subsystems that eliminate the need for the clock synthesizer. This class of RF board works correctly only on the 102-73181-10 boards.

This module is missing connections to function correctly.

3.15.12 102-73181-24 SA818 1W RF transceiver

SA818 or DRA818 RF module (deficient design).

This daughter board is a minor upgrade to the 102-73181-22 board.

Add capability to switch between HI and LO power.
Add DC block to audio input to DRA818.
Add D2 and DS2 to allow eliminating regulators.
Change VR1 to 4.2V regulator (eliminate adjustable regulator). Also much lower cost.

Incorrect pinout for the 4.2V regulator. This module is missing connections to function correctly. par

3.15.13 102-73181-34 SA818 1W RF transceiver

SA818 or DRA818 RF module (deficient design).

This daughter board is a minor upgrade to the 102-73181-24 board.

Split the PTT (pusk-to-talk, the transmit enable line) and PD (power down) functions such that the zNEO control these signals with separate pins. The zNEO now has complete timing control.

Add attenuator to the output so we can make an RF daughter board that doesn't produce so much RF power.

Incorrect pinout for the 4.2V regulator. This module is missing connections to function correctly. par

3.15.14 102-73181-36 SA818 1W RF transceiver

This daughter board is the upgrade that gets if right!
The schematic is on page 10. A a board image may be found on page 11.

This final revision breaks the SA818 **PD*** net from the **PTT*** to allow the software control of the timing required to wake up.// The 102-73181-10 board also seperates the daughterboard power contrl from the **PTT*** net. Again, to allow the software control of the timing .
These changes finally brought the SA818/DRA818 module to life.

Power levels seem to be lower than advertised, bu that should not present a problem for this application.

4 Operation

Notes on the physical operation of the FOX Transmitter.

4.1 Power

The device is powered in the field using a battery. An LR22 9V may be expected to run the unit for several hours. There is enough room in the housing to make use of a 6-cell AAA battery. Note that a set of 6 AAA alkaline batteries are cheaper to purchase than a single LR22. The 6-cell pack should last considerably longer.

Either of these connect to the circuit board using a 3 pin locking connector.

It should also be possible to also use a 5 cell arrangement with a 2-cell and a 3-cell battery holder to provide 7.5V. Either way should provide more than 8 hours of operation.

Revision 1 boards (*102-73161-12*) are also provisioned with a jumper that takes power from the USB bus when connected to a host machine. This can be used to leave the transmitter powered for several hours to charge the battery for the TOY clock.

Revision 2 boards (*102-73161-25*) change the 5V regulator to a switchmode device to further improve battery life.

The switchmode regulator allow the use of higher voltage batteries without the accompanying heat. Switching to lithium chemistry batteries would allow extended operation or operating a higher power levels.

4.2 Antenna

The output matching network and output filter assume a 50 Ohm antenna system. Power levels are low enough that matching should not be critical. This is probably a good candidate for a J-pole made from 300Ω twinlead.

An impedance mis-match will affect the performance of the filter and the radiated power.

The board may be built with tip jacks in place of the BNC connector. These are intended to be used with a simple wire dipole antenna. Cut the wire to length and solder into a tip plug. The plug may then be attached in the field.

Rubber Ducky From Amazon:
NAGOYA Dual Band UHF/VHF
Super Soft Flexible Two Way Radio
BNC Connector Antenna 144/430MHz
Orange(2packs)

HYS-771 144/430 MHz
Dual-Band High Gain Antenna
BNC
15.6 inches
VHF/UHF Radio

4.3 Jumpers

Table 4: Jumpers

Ref Des	IN	OUT	Description
JP2	*MAS=*		MASTER Jumper
JP3	*TEST=*		TEST Jumper
JP4	*USB*	*Battery*	USB Power Configuration
JP5	*Devel*	*Deploy*	Code LED
JP6	*Devel*	*Deploy*	Code Buzzer
JP7	*SA818*	*SI5351*	5V daughter board power bypass

4.3.1 Jumper JP2/JP3: MASTER/TEST

Sheet 4.6E/Sheet 3.2A
Installation of this jumpers control how the fox transmitter starts up. All 4 combinations produce results.

Table 5: MAS/TEST Jumpers

JP2 MAS	JP3 TEST	Files Run	Description
OUT	OUT	INI= ANN=	System Identity Announce Message
OUT	IN	INI= TEST=	System Identity Test commands
IN	OUT	INI= MAS=	System Identity Master commands
IN	IN		no commands are run (fault recovery)

In normal operation, i.e. no jumpers in these two positions, we run the initialization commands and then send the announce message. The announce message is intended to let the hunt operator know that the transmitter is functioning as it is placed for the hunt.

When jumpers are in place, the announce message is replaced by the **TEST=** commands (JP3) or the **MAS=** commands (JP2).

Installing both jumpers places the system in a fault recovery state. This skips all commands from the FRAM to allow control of the uninitialized system from the serial port.

4.3.2 Jumper JP4: USB Power

Sheet 3.4A
USB Power Bypass. Installing this jumper powers the station from the USB bus (assuming that J2 and U6 are populated).

4.3.3 Jumper JP5: LED

Sheet 2.8D
Enables the D2 LED.
Leave this jumper out when operating in the field, you can't see it and there is no point in wasting the power to drive it!

This jumper and the associated MOSFET and LED may not be populated.

4.3.4 Jumper JP6: BUZZER

Sheet 2.9D
Enables the thoroughly annoying BZ1 buzzer.
Leave this jumper out when operating in the field, you'll give your position away!

As with the JP5 jumper, this jumper and the associated MOSFET and buzzer may not be populated.

4.3.5 Jumper JP7: DB Power

Sheet 5.4C
Installing this jumper bypasses the 5V power switch that supplies 5V to the RF daughter board.

This jumper is necessary when using the SA818/DRA818 RF subsystem with the 102-73181-5 artwork revision. The 102-73181-10 artwork revision has a provision to control the power switch, U81, from a separate port. This has the potential to reduce power consumption slightly by removing power from the DRA818/SA818 module. The 102-73181-10 artwork can use the separate power control as a fail-safe to de-activate the DRA818/SA818 module. The DRA818/SA818 module seems to require a fairly bit of time to recover from a power-down.

4.4 Resistor Jumpers

Table 6: Resistor Selection

Ref Des	**1-2**	**2-3**	**Description**
R26	*5V USB*	*5V Local*	FT232RL VCC Voltage

4.4.1 Resistor Jumper: R26

Sheet 3.5B
Normally the FT232R is powered by the USB bus so the host connection doesn't disconnect during software development.

During operation, powering from the USB removes the current draw of the FT232R from the battery.

This position will **not** be populated if the FT232RL is not present.

4.4.2 Resistor Jumper: R5 & R6

Selects the pin on the programming header connected to the zNEO.
Always install R5, leaving R6 unpopulated.

4.4.3 Resistor Jumper: R15 & R18

102-73181-10: Install as indicated on the silkscreen (vertical).

This jumper pair connects the zNEO port 1 pins to the FT232RL serial pins and to the U1 buffer for the TTL-232R-3V3-AJ cable.

4.4.4 Resistor Jumper: R1 & R9

102-73181-10: Install as indicated on the silkscreen (vertical).

This jumper pair connects the zNEO port 0 pins to the inter-board jumpers to the DRA818/SA818 daughter board (the daughter board also has resistor positions to accomplish the same function).

4.4.5 Resistor Jumper: R64 & R22

102-73181-10: Install as indicated on the silkscreen (vertical).

This jumper pair connects the other side of the U1 buffer to the physical connector for the TTL-232R-3V3-AJ cable.

4.4.6 Resistor Jumper: R68

102-73181-10: Install in position 2-3 for proper DRA818/SA818 operation.

This jumper position seperates the power switching function from the transmit enable function.

4.5 Time

Timekeeping in the processor subsystem is present to allow synchronization of multiple transmitters. To accomplish this a reasonably accurate clock system is required, but it does not need to track from any particular epoch. As long as all the transmitters are using the same starting point the scheduling algorithm will function correctly.

A truncated time is entirely adequate. The Linux based utility that is used to load the FRAM uses a truncated Linux time to update the TOY clock.

Internally the zNEO uses the clock time and truncates it to seconds of day (using modular arithmetic). Having the internal time truncated to a day makes the arithmetic a bit easier to deal with. The algorithm works with any timetag that is aligned with the start of the day.

4.5.1 Time Network

The time synchronization network function that appeared on the J4 connector has been deprecated. The J4 connector function on the 102-73181-10 boards is now the zNEO host control port. The J4 connector on older revisions should not be connected to anything, it is assigned to the DRA818/SA818 driver.

This moves the connection point outside the case on the 102-73181-10 boards so no disassembly is required when updating the station time.

4.5.2 Time Synch Procedure

Time synchronization on the 102-73181 boards may be accomplished through J4 or J5, whichever is installed. The 102-73161-25 boards are controlled through the J5 connector.

Typical command line to update the stations TOY clock:

```
    /home/wtr/Radio/halo_term/fox_simple -SFOX7  -t10
    /home/wtr/Radio/halo_term/fox_simple -STACH  -t10
```

Typical command line to update the FRAM and set the time:

```
    /home/wtr/Radio/halo_term/fox_simple -SFOX7  -t10 -ffox7.fox
    /home/wtr/Radio/halo_term/fox_simple -STACH  -t10 -ffox20.fox
```

Examples above are commands used by the author to access the 102-73161 units (FOX7) of the 102-73181 units (TACH). The 102-73161 units have unique names as the USB controller resides on the fox circuit board. The 102-73181 units move the USB UART to an external cable (only one USB device for all of them) to reduce unit cost and simplify access to the unit.

4.6 Developing A Message Sequence

A short discussion of how to go about developing a set of message traffic for a *fox group*.

First off, when talking about a *fox group*, we are simply referring to a group of fox transmitters all operating on the same frequency and a synchronized schedule. This set of transmitters must have matching scheduling periods and correctly staggered scheduling offsets so that they can be individually heard. In other words, we typically don't want them ever to transmit concurrently.

Second, note that they all could operate with the same sequence with just a few unique commands (i.e. station nickname and scheduling offset).

In the following discussion, the presence of the TEST and MAS jumpers affect which setup files are run.

When a fox transmitter is first commissioned, or when the FRAM has been erased, you can easily observe which files are searched for through the serial port.

The TEST= file should be very limited or empty to provide a catastrophic error recovery path. With the TEST jumper in place, only the TEST= commands will be run.

4.6.1 Identification and basic voice clips

The stations, although sharing a common callsign, may want to be individualized. Typically with a *stroke* and a *number*: KA0AAA/1 and KA0AAA/2, etc:

The signon message is built as follows:

```
sprintf(buffer, "e.. CQ CQ CQ DE %s ", fox_config.CallSign);
```

note that the nickname is not sent as part of the standard signon message that is generated by the **BEGN** command.

The station nickname should be unique to allow the hunters to properly (quickly) identify the station. This form of station identification requires a **CODE** or a **TALK** command to send the saved nickname out using code or voice.

The station callsign and nickname should be saved in waveform memory to be accessed by the **TALK** command. Voice identification will be desirable for those that can't easily read code.

The set of battery reporting voice clips should be included to allow the unit to verbally report battery condition throughout the hunt. Here is a sample load of the file fragments that will be used:

```
esav TALK=BATTI 44 4140 4K
esav TALK=BATTV 4268 4374 4K
esav TALK=REG5 8748 5016 4K
esav TALK=POINT 13868 1316 4K
esav TALK=HZ 15276 2290 4K
esav TALK=KHZ 17708 3054 4K
esav TALK=MHZ 20908 3130 4K
esav TALK=N0 24108 2578 4K
esav TALK=N1 26796 1730 4K
esav TALK=N2 28588 2114 4K
esav TALK=N3 30764 1858 4K
esav TALK=N4 32684 1858 4K
esav TALK=N5 34604 2126 4K
esav TALK=N6 36780 1672 4K
esav TALK=N7 38572 1870 4K
esav TALK=N8 40492 1474 4K
esav TALK=N9 42028 2276 4K
esav TALK=MAMP 44460 3586 4K
esav TALK=VOLTS 48172 2922 4K
```

The last two clips are application dependent, the filenames will match the callsign and nickname saved in to waveform memory.

```
esav TALK=KA0AAA 51244 5620 4K
esav TALK=FOX20 63148 6130 4K
esav TALK=FOX21 65148 6130 4K
esav TALK=FOX22 67148 6130 4K
esav TALK=FOX23 69148 6130 4K
esav TALK=FOX24 71148 6130 4K
```

This example assigns the same callsign to all units and loads all the nicknames into waveform memory. The waveform load image will the be the same on all units and will require a flash device of adequate size to hold all of the voice clips.

4.6.2 Initialization

Once the audio waveform data is loaded into waveform memory (in the form of an INTEL HEX file) and the audio filesystem is loaded into FRAM memory (as a set of esav TALK... commands) we can proceed with initialization.

The **INI=** file runs following reset *unless* both the **TEST** and **MAS** jumpers are in place. The **TEST** jumper, when in place, triggers the **TEST=** file after the **INI=** file runs. The **MAS** jumper, when in place, triggers the **MAS=** file after the **INI=** file runs.

This is the **INI=** file that is run at startup.

```
esav INI=NAME FOX20
esav INI=CALL KA0AAA
esav INI=TIME
esav INI=EPOC −5.0

esav INI=CONF SA818
esav INI=CONF T1=2500
esav INI=CONF T2=150

esav INI=FREQ 144.150
esav INI=MODS,S0,300,0
esav INI=MODS,S1,300,60
esav INI=MODS,S2,300,120
esav INI=MODS,S3,300,180
esav INI=MODS,S4,300,240
```

This initialization sequence sets the unit callsign, nickname, time and timezone.

We then configure the RF subsystem, selecting the RF hardware we are using and changing any required operating parameters. In this example, we extend the time allowed for the SA818 module to stabilize (T1) before we start sending serial commands to configure it when starting to send a message over the air. We also allow the RF to stabilize a bit using the T2 parameter.

The SA818/DRA818 modules have little in the way of configuration control we are interested in. We are exclusively transmitting, and with the 102-73181-10 revision boards, we will remove power from the module when it is not in use.

One timing parameter that may need adjustment is the T1 state delay. Selecting SA818 or DRA818 sets e default value of 2000 mSec which seems to work when the module is continuously powered. The 102-73181-10 revision boards require some additional time for the module to be ready for commands following power-up.

Frequency selection simply requires we tell the system what to use. In out example we have selected the announce frequency of 155.150MHz.

We also setup the master schedule, specifying the schedule each of our stations will use when operating. There is no need for any unit to know all the schedules, we do it simply for our convenience.

4.6.3 Announce

The announce message is sent when the unit is powered on to provide a sanity check for the hunt operators during setup.

```
esav ANN=TONE 1.0
esav ANN=CWPM 20,-1,-1,-1,-1
esav ANN=BEGN
esav ANN=TALK <CALL>
esav ANN=TALK <NAME>
esav ANN=BATC EV 7.2
esav ANN=BATC EI
esav ANN=BATV V
esav ANN=BATV I
esav ANN=DONE

esav ANN=FREQ $2
esav ANN=TONE 1.6
esav ANN=CWPM 15
esav ANN=STAT
```

Just a few things you will want to keep track of in the setup. The setup is broken into multiple sections: INI=, ANN=, TEST=, and MAS=. Which of these gets run is controlled by the TEST and MAS jumpers (see section 4.3.1 on page 48).

The (setup) FREQ command sits in the INI= file to force a frequency selection to occur no matter the state of the TEST and MAS jumpers. We set the target operating frequency after we send the initialization message.

Next set the code speed up to keep the message length somewhat reasonable and turn the transmitter on (the **BEGN** command). The **BEGN** command will send of a CQ with the callsign we have set earlier.

Once the code traffic clears we call up the callsign and nickname in waveform memory and set it out (verbally) to let the operation know he has planted the desired fox station.

We now go on to battery reporting, using two battery reporting commands.

> The **BATC** command report in code. In our example above, we ask for *Encoded Voltage* with a trip point of 7.2 volts. We get a bit of code that either reports a voltage above the trip point ("BATC HI HI TTTTTTT EEE") or a low voltage condition ("BATC SOS SOS TTTTTT EEEEEEE").
>
> The battery voltage is encoded as a series of DAHs and DITs representing the units and tenths. This reduces to listening for either HI HI (.... .. ..) or SOS SOS (... — ... ... — ...) and counting longs and shorts.
>
> The following **BATC** command (still code) reports the measured current draw. This will be measured when the command is executed so it reflects the power draw with the RF section active.
>
> It is encoded much like the voltage section, starting with ("BATC III TTTTTTT EEE"). The tens position is the string of Ts and the units position is the string of Es.
>
> The **BATV** command works in much the same manner as the **BATC** command other than sending its result using voice files.
>
> The first command (**BATV V**) reports voltage to two decimal places. and the second (**BATV I**) reports current in milliamps.

As this is simply a message to report that the station is alive and report battery condition when we power on, we send the **DONE** messsage and shut down the RF subsystem.

4.6.4 Next

At this point, after the **DONE** command, we reset the operating frequency to our assigned frequency and begin running our schedule.

The **TONE**, **CWPM** commands provides a baseline code *personality* if the schedule is missing these setup commands.

The **STAT** command is useful only when connected to a host. When deploying for the hunt, it has no practical effect.

4.7 Deployment

Once time synchronization has been achieved on the bench the units may be powered off to conserve battery life. They can then be powered on when deposited in their hiding locations. Following initialization of a few seconds, the units run the **ANN=** sequence which should report the units callsign and name on a base frequency shared by all units being deployed.

Pay attention to the startup (power-on) message traffic, listening for the battery message.
This battery message gives an idea of when units are soon to go silent. If the trip point is correctly selected, the unit will send an "SOS" in the battery message when the battery voltage falls below the trip point. When the battery voltage is good, a "HI" message is sent.

A final note on operation in the field:
Your only effective device control in the field is the power switch. There are no write activities that occur without sending commands. If you have correctly formed the schedules, that is to say there are no commands in the schedule that write to the FRAM or FLASH, then powering off the unit will not corrupt non-volatile storage.

This leaves you free to arbitrarily remove power from the transmitter at any time. If there are incorrect transmissions, simply shut the unit off.

4.7.1 Deployment using multiple frequencies

Consider a hunt operating multiple transmitter groups. When operating more than one set of transmitters it should be obvious that we will operate on multiple frequencies.

To make station deploy at the beginning of the hunt as seamless as possible, consider starting the transmitters all on one *startup* frequency, say 144.150MHz, and switching over to an operating frequency once the station is setup. A setup something like:

```
esav INI=CALL W0JV/20
esav INI=NAME FOX20
esav INI=TIME
esav INI=EPOC -6.0
esav INI=CWPM 25
esav INI=CONF SI5351
esav INI=CONF CLK0
esav INI=FREQ 144.150
esav INI=MODS S0 300 0
esav INI=MODS S1 300 60
esav INI=MODS S2 300 120
esav INI=MODS S3 300 180
esav INI=MODS S4 300 240

esav ANN=BEGN
esav ANN=BATV V
esav ANN=BATV I
esav ANN=BATV R
esav ANN=DONE
esav ANN=FREQ 144.450
esav ANN=RUN0,S2
esav ANN=STAT
```

The setup proceeds more-or-less normally through line 8, although we specify the startup frequency (in this example 155.150MHz). Lines 9 through 13 setup the operating schedules for all stations in this group. We will also load all schedules in the group into each station, changing only lines 2 and 21 (although we don't show any of that here). This is simply to make the management task a bit easier.

When the unit is switched on, the **INI=** commands are executed to setup the station (lines 1 ..13).

Lines 15..22 are the announce message that runs right after the INI= commands.

z The **BEGN/DONE** commands will, as expected, enable the radio and send out the requested traffic.

We will get a **CQ** along with a station callsign from the **BEGN** command, a vocal battery status report from the three **BATV** commands, and finally a station callsign followed with **SK**. The transmitter will be shut off, exactly as expected from the **DONE** command.

Line 20 will then change the operating frequency to that which the group of stations are operating on. At this point the next **BEGN** command will transmit on the new frequency unless the S*= sequence changes it.

The **RUN0** command (on line 21) enables one of the 5 schedules we will have stored in the FRAM (some time before the hunt). The station is now ready to participate in the hunt.

The textbfRUN0 command is placed here for clarity. In practice it may be located anywhere after the **DONE** command. It need not occur in the FRAM file system adjacent to the other **INI=** commands.

So when loading the 5 station group, the commands are loaded from the common file to each station. The unique **RUN0** commands may then be manually added following the file load.

When looking at contents of the FRAM, the **INI=RUN0,Sn** command will be displaced from the rest of the **INI=** commands, but the file system doesn't particularly care.

The **STAT** command on line 21 is present only as a debug aid (or as a sanity check) for the operator when loading the station. When the station is connected to the host system, the setup can be verified without having to manually enter a command. The time required to run the **STAT** command is about one second. The time to display the status reports depending somewhat on how the configuration affects the number of lines of information that is displayed.

4.7.2 Dropping Stations at the Hunt

If you've done your homework the night before, all that is required to setup for the hunt is to power-on the station as it is dropped off at its operating location.

Your H.T. will be tuned to the startup frequency so you can hear the station report to know that things are operating as intended. Pay attention to the battery report (vocalized by lines 16..18) to see that battery voltage is adequate (above 7.w volts in our example). One lower power units we can also include a battery current report to see that it isn't excessive (expect less than 50 mA on a fresh battery).

The placement order has no effect on hunt operation. The stations use the TOY clock to time their transmissions.

4.8 External Transceiver

The board may be configured to operate an external transceiver using the J6 connector. Typically the ICS525/ICS307/SI5351/DRA818/SA818 would simply not be installed for this application, but this is strictly not necessary if the antenna connector is properly terminated to prevent stray RF.

The -25 artwork makes use of an RF daughter board that may be removed to prevent RF from escaping.

The pinout is found in table 3.9 on page 29.

Mating housings for the various configurations:

Table 7: J6 housing reference

Vendor Number	**DigiKey Number**	**Vendor Name**	**Description**
102387-1	A25901-ND	TE Connectivity AMP Connectors	10-pin housing
102387-2	A25902-ND	TE Connectivity AMP Connectors	14-pin housing
87756-4	A25969CT-ND	TE Connectivity AMP Connectors	CONN SOCKET 22-26AWG CRIMP GOLD
87523-5	A25993CT-ND	TE Connectivity AMP Connectors	CONN SOCKET 22-24AWG CRIMP TIN
0901420010	WM8037-ND	Molex	10-pin housing
0901420012	WM8038-ND	Molex	12-pin housing
0901420014	WM8039-ND	Molex	14-pin housing
0901190110	WM2580CT-ND	Molex	CONN SOCKET 22-24AWG CRIMP GOLD
0901190109	WM2581CT-ND	Molex	CONN SOCKET 22-24AWG CRIMP TIN

A mating housing from TE Connectivity (102387-1) with crimp contacts (87756-4) may be used to fabricate a cable to connect to an external transceiver.

4.8.1 VCMO_TONE

Square Wave from timer in zNEO.

This signal is buffered by a tri-state driver that is disabled when no tone is required. The termination network pulls the voltage when the buffer is tri-state between tone bursts..

4.8.2 FILT_TONE

Filtered VCMO_TONE.

The VCMO_TONE signal is run through a simple RC filter to attenuate harmonics and presented on this pin.

4.8.3 AC_TONE

FILT_TONE witch DC isolation. Ground centered.

The FILT_TONE is passed through a capacitor to eliminate the DC offset and presented on this pin. A large value resistor keeps the DC level near ground.

4.8.4 PTT

N-channel MOSFET switch to ground when transmitting.

The ZXMN3A01 device installed on the circuit board is a 30V device. Although rated to pass 1.8A, the circuit board does not provide traces that can carry this current. Limit current draw to less than 100mA.

4.8.5 VBATT

Unswitched battery voltage.

The FOX transmitter may be powered through this pin to eliminate the internal battery. The power switch is still used to switch the unit on.

4.8.6 V9.0

Switched battery voltage.

If the battery provides adequate current and life, the external transmitter may be powered from this pin.

4.8.7 SWITCH

Panel switch input.

Same function as 102-73176 pin with 4.7K pull-up to V3.3.

4.8.8 PHOTO_CELL

Photo Cell input.

Same function as 102-73176 pin with 220K pull-up to V3.3.

4.8.9 GNDP

Transmitter ground.

4.9 CHRP: Chirping

A special modes was added to the V3.50 software, a *"chirp"* mode. This is **not** a radar chirp mode where the audio frequency or the carrier frequency sweeps. Rather this emulates the operation of a tracker such as that that would be used to track wildlife.
Chirp here referes to a short period of carrier plus audio followed by a period of quiet.

The command to initiate this operating mode is the **CHRP** command along with a few parameters that control its operation. See section 9.2.23 on page 124 for a command description.

This is a discontinuous mode where the carrier is disabled between chirps. Setup parameters control the audio modulation frequency, the timing of the chirp, and a repeat count to simulate continuous operation.

As we are operating within the limits of part 97, we are required to identify on a regular basis (every 10 minutes) so we must tailor the time the emulation is active using the **CHRP** parameters.
The **CHRP** command must occur between a **BEGN** command and a **DONE** command. In this respect it operates just like a **CODE** or **TALK** command. The **BEGN** and **DONE** provide station identification in code at the begining and end of our message. When using multiple **CHRP** commands you may find it necessary to send out the station callsign (using **CODE <CALL>** or **TALK <CALL>**).

4.9.1 R68 in alternate position

This is the position that FOX21..FOX26 have the power control resistor strapped to. The resistor must be in this position when using the DRA818/SA818 modules, for carrier control to work as expected.

4.9.2 R68 in primary position

This position uses the **TX_ENA** net to switch on the U81/U91 daughterboard power switch. This position requires **not** using the DRA818/SA818 module. This is compatible with any of the small RF amplifier daughterboards.

In this configuratio the **TX_ENA** net is used to remove power between chirps.

When R68 is in this position, timing for control of the DRA818/SA818 module is to short for proper operation.

4.9.3 CONF DB_PWR

The older RF modules may be used for chirping by setting the **DB_PWR** configuration flag. This allows the chirp handler to switch daughterboard power on and off using U81 on the motherboard (controlled by the DB_PWR net).

4.10 Status and Configuration Reporting

Status and Configuration reports have become more extensive in this release. The **STAT** command generates over 20 lines of information and the **CONF** command is respoinsible for over 40 lines.

The **STAT** command will give a summary of the current system configuration and the state of the battery. If you have caused the transmitter to be enabled, the voltage and current measured during tr4ansmit will also be displayed. The battery monitoring configuration is hardware revision dependant, the RF module configuration must be set correctly collect battery information.

CONF command dumps the configuration bits (a 32 bit longword array), showing which are set. There are also settings for the RF subsystem that are also controlled here.

4.10.1 Status Report

- Software status
- TOY clock sstatus
- Update Flag
- Jumpers (TEST and MAS)
- External memory devices
- Battery status
- Analog channels
- UART buffer status

4.10.2 Configuration Report

- RFGEN
- RADIO
- CWISR
- AUDIO
- SYNTH
- ANALOG
- DEBUG
- SI5351
- BMON
- VOICE
- T<n> timing

5 Assembly

Assembly Hints

5.1 Board Inspection

Verify the circuit board is undamaged.
Soldermask and silkscreen should be undamaged.

5.2 Parts Ordering

The parts in the project were ordered from DigiKey and Mouser.

The most convenient file to use for ordering is the *102_73181_10.DigiKey_bom.csv* which can be used to load a DigiKey cart up using the BOM manager. There will, no doubt, be some parts that will need attention due to becoming obsolete or simply not being currently in stock. Specific parts are non-critical as long as 1% parts are used for resistors and BP/NPO parts are used where values are specified in *PF*. Use BX/X7R for remaining ceramic capacitors.

Although the standard package used for capacitors and resistors is the 0805 so that part markings are somewhat legible, there are quite a few parts that are 0603 packages to improve performance (bypass capacitors) or simply to reduce footprint.

5.3 Parts Labels

A printable file is generated to print labels on 10-up or 14-up label stock. Print the *102_73181_10.lbl.ps* file if the parts labels make assembly easier for you.

The *Idx #* number matches up in all the parts lists.

5.4 Parts Placement

Print the *102_73181_10.mbr.ps* file which is a comprehensive parts list grouped by part value. The *Component MIT* column has the part location on the circuit board (expressed in inches). The zero reference is the lower left corner of the board as viewed from the top.

The *Idx #* column number matches up in all the parts lists. Later board layouts, i.e. the *102_73181_5_pwb.pdf* drawing, has a scale referenced to the board zero point on both sides of the board.

Load surface mount resistors and ceramic capacitors first. These are the low profile parts that do not interfere with larger parts.

Add tantalum caps and inductors next.

Active devices can be installed next with the exception of the 5V regulator which has a rather high profile. Note that once you begin installing active devices, careful static discipline must be observed or damage may result. Always handle the boards on a static dissipative surface, use dissipative wrist straps and always store the boards in static bags.

Finish off with the large parts with the exception of the battery, which we will install after initial testing.

Clean your board. 99% ethanol or 99% isopropanol works well. 70% isopropanol (rubbing alcohol) should be avoided.

5.4.1 5V Regulators

Verify the pinout and orientation of the 5V switching regulator when installing this part.

There are two variants of the vertical mount device that can be obtained from DigiKey. Verify which pin is the input pin and orient the regulator with that pin towards the top edge of the board. There are extra pads under the device to allow rotating 180°.

5.5 Daughter board Mounting

The RF daughter board is mounted using two or three 12mm spacers and pan head machine screws.

The two spacers along the bottom edge are located between the mounting screws. These are both drilled for 3mm fasteners. The third hole, located to the right of the zNEO chip is drilled for a 2.5mm fastener.

The 12mm height fits well with the connector heights.

We can also provision with #4 and #3 fasteners using 0.472" or 0.500" spacers. Note, however, the #3 spacers are very difficult to find, hence the move to metric dimensioned fasteners.

Also keep in mind that the 0.472" dimension is 12mm, so choose this dimension to maximize interchangeability.

Daughter board connectors are soldered with the boards mounted together using the target hardware. This keeps the connectors properly aligned to allow free interchange of the daughter boards.

The motherboard would nominally have the socket connectors (Sullins PPPC031, PPPC041, and PPPC061) installed on the motherboard. The daughter board pins are cut from a strip (Sullins PBC03SACN or PBC03SADN) with a mating length of 0.230" and a tail length of 0.320" or 0.420". Excess length being trimmed after soldering.

Assemble the mating parts, insert them into the board pair and install the fasteners. Press the connector assembly toward the motherboard and solder.

The prototype build required both a 102-73181-26 and a 102-73161-29 board to get all the sockets installed on the motherboard. When building multiple transmitters, pin headers can be mounted on spare 102-73181-26 and 102-73161-29 boards to speed up installation of the sockets on the motherboard.

Multiple RF amps can be dealt with in the same manner, using a spare 102-73181-10 board with socket headers installed to mount all the pins on your set of RF amplifier boards.

6 Haywires

4117 Several wiring modifications may be made to the boards to improve function. These updates require adding *haywires* and possibly some passive parts.

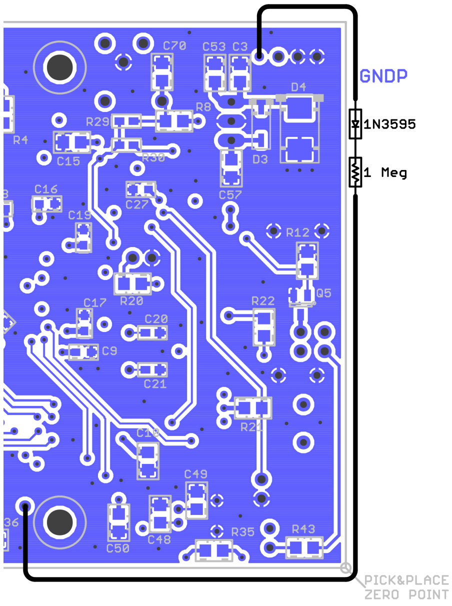
6.1 Audio/Voice

This wiring hack is available to enable audio capability on the -25 board. This consists of a resistor between U1-45 and TP4. Software in the zNEO explicitly configures U1-46 as an input to allow U1-45 and U1-46 to be shorted. TO apply this hack obtain a 1/20W 750Ω(680Ωor 820Ωwill also work) resistor and suitable sleeving.

4141 Estimate the position of the resistor between U1-46 and TP4 and cut sleeving to an appropriate length. Tin the lead ends and the slide the sleeving into place. One the U1 end of the resistor, bend one side of the resistor to 90° and trim it to fit U1 (about 0.025"). Solder the prepared end to U1-45 and U1-46 by holding the resistor body vertically. You can then bend the resistor down to the surface of the circuit board and fit the other end to TP4 and solder.

6.2 TOY Clock Battery Maintenance

4150 This hack adds a charging circuit to keep the TOY clock battery fully charged when the transmitter is left switched off with batteries left in place.



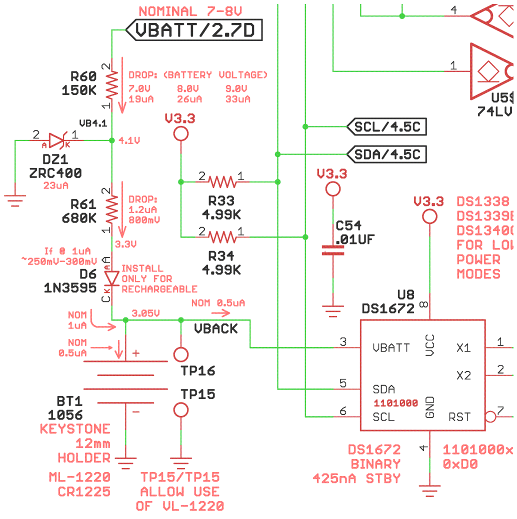
For this we add a 1N3595 low leakage diode in series with a 1M to 4.7M resistor between the battery connector *J2* and the clock battery *B2*. The diode band *cathode* is towards clock battery *B2*.

This provides about 1uA of current from the main battery, which is assumed to be a 6-cell alkaline pack, to the backup battery and the TOY clock. The TOY clock requires about half of this (about 500nA) to keep its 32KHz oscillator running with the remaining 500nA being driven through the backup battery. 4181

This image is of the 102-73161-25 artwork, but earlier revisions are similar. Connect the top of the diode/resistor set to the 12V pin on the battery header (on the -25 artwork there is a convenient via that may also be used, as shown at the right). The other end is connected to the the via on the positive lead of the coin cell.

The diode/resistor is insulated with heat-shrink and hot-melt glued to the backside of the circuit board.

This function is present on the 102-73181-5 artwork. albeit in a slightly more sophisticated form. The 102-73181-5 circuit provides better current regulation as the main battery voltage falls.



R60/DZ1 form a 4.0V regulated supply that is dropped across R61/D6 to supply current to the coin cell and the DS1672. This is intended to keep the battery (BT1) floating at about 3 volts.

The 1N3595 was chosen for very low reverse leakage current. A 1N4148/1N914 may be substituted.

4217 The 102-73181-10 circuit board is provisioned to accommodate either the ML-1220 battery (no longer in production) or a 12mm coin cell holder. The 12mm coin cell holder fits a CR1220 or BR1220 lithium cell. The charge circuit limits the current passed through to the battery and TOY clock to on the order of about a microamp, keeping it compatible with the use of primary (non-rechargeable) cells.

The main battery, then, should supply current to the TOY clock keeping the backup battery stable for a long period of time.

7 Commissioning

Initial Testing

7.1 Basic Tests

Some sanity checking to look for cold or missing solder joints and shorts.

Voltages

Apply power and verify the 5V and 3.3V rails are at the correct voltage. Two ground pads are located near the static symbol.

Check positive end of C4 for battery voltage.

Check 5V regulator at vias near left side of C5.

Check 3.3V regulator.

102-73181-5: at vias below and to the right of VR2.

102-73181-10: at the tab of VR2.

Software Load

Load software into the zNEO (programming header J3).

Attach programming cable to J3 and use the ZiLOG tools (ZDS-II/zNEO) to load the operating software into the Z16F2810.

Attach a USB cable to J4 and reset the system looking for a prompt through the serial/USB connection. Communication settings are 19,200 bits/second, 8 bit characters and no parity.

Reset using reset button, REST command, or by power cycling. Assuming you are connected to a host, expect something like the following.

Listing 1: Startup

```
sts01,00* ****************************************************************
sts01,01* KC0JFQ  FOX Transmitter  V3.54
sts01,02*   Z16F2810AG20EG
sts01,03* Tools Ver: 20230510  <ROM:26074 EROM:88852 Flash:114926>
sts01,04* Flash Prog 04.25.03 MB85RS256TY    FLASH_FRAM       Fujitsu   256K-bits    1024-records CMD3
sts01,05* Flash WAVE 20.71.14    M25PX80    FLASH_PAGE        Micron  8192K-bits     125-seconds CMD4
sts01,06* ****************************************************************
```

Using the *fox_simple* utility to load the operating program into FRAM and the audio file system int FLASH.

You may observe progress of the load operation through the serial port.

7.2 RF Tests, SI5351

Some sanity checking to look for cold or missing solder joints and shorts.

Attach an RF amplifier daughter board (either 102-73161-29 or the 102-73161-28 board)

Configure for the SI5351 and the daughter board in use.

Connect a dummy load or power meter.

Send the **BEGN** command and verify RF is correct frequency and amplitude.

7.3 RF Tests, DRA818

More sanity checking to look for cold or missing solder joints and shorts.

Attach an RF module daughter board (102-73181-36).
Install the low power jumper JP2 to limit the DRA818/SA818 output level.

Connect a dummy load or power meter (use an external attenuator if necessary to handle an output level in excess of one watt. The authors experience with the DRA818/SA818 indicates power levels of less than 200mW with JP2 im plae on the 102-73181-36 daughterboard.

Configure for the DRA818 and send the **BEGN** command.

7.4 Install backup battery

After the unit checks out and is ready to install in its housing, the backup battery can be installed on the board. We have deferred to this point to avoid discharging the battery during checkout when the bare board would be stored in a static bag.

Place Kapton tape under the battery to isolate it from the vias that occur under the battery.

Use the schedule loader to set the TOY clock.

7.5 Loading FRAM and FLASH

Use the host tools to load to load the operating configuration commands and waveform date into FRAM.

Although order is not typically important, try to load the waveform data first followed by the configuration commands (and schedule) and then any directory records required by the waveform image.

Keep in mind that the *ERAS DEV* command clears all of FRAM and will require reloading the waveform data. V1.54 adds a *ERAS CMD* to clear only the first 1024 records. Around V3.50 an additional qualifier was added to add flexibility: *ERAS HALF*. This is provided to osupport the 102-73161 units that have no FLASH memory. You may find that using very short audio clips speeds sequence development. Once your operating sequence is working, substitute back in the full audio waveform files.

The directory file produced by the Audio File Utility *pwm_audio_util* may also be loaded using *fox_simple*, although it may be faster to simply copy&paste them for small files.
Also keep the FRAM (eras) and FLASH(hera) commands straight! Reloading FRAM is usually fast due to the small number of records store there. FLASH is much larger and typically gets bulk erased.

8 Software

Breakdown of the software subsystems.

8.1 Scheduling

The basic scheduling algorithm used to coordinate message transmission has roots in the clocking methodologies used in the Voyager and Cassini spacecraft. This methodology is carried forward into the WAVES instrument on the JUNO spacecraft. In addition, two Earth orbiters, the AmSAT Fox1D spacecraft HERCI instrument and on the HALOSAT spacecraft.

With this impressive heritage, let us dive into the scheduling algorithm and see how we make use of it to control message delivery.

8.1.1 Goals

The problem this method addresses is how to schedule multiple activities or events without the need for communications between these independent events. All that is common to everyone is a time counter that the same everywhere. So as we continue, we operate under the assumption that the clocks running in all the FOX Transmitters are operating with identical epochs. Since all units operate with the same software, clock formats are identical. There is some magic (well, magic until details are revealed in later sections) that keeps all the clock synchronized.

In the case of our FOX Transmitter, we want to describe or specify a schedule for transmitting messages that allows multiple transmitters to share a common channel (frequency), with each transmitter getting some time to speak up and be discovered by the fox hunters.

Start by thinking of all FOXes operating with the same cycle time but all transmitting at different time in the cycle. This is a good way to start to understand the scheduling algorithm, but there is considerable flexibility in the way we describe schedules.

8.1.2 Fractional Seconds

Now let us consider what a meaningful time granularity is going to be for our purposes. For background, the scheduling granularity in the Cassini/RPWS instrument was 125mS and all succeeding instruments (JUNO/WAVES, Fox1D/HERCI and HALOSAT) has been 40mS. These periods are driven by the instruments operating cadence.

For our FOX Transmitter we will adopt a scheduling granularity of 1000mS (that is to say one second). The scheduling method used in all FOX transmitters is the same and relies on all units having the same time.

We can describe any arbitrary schedule that starts on a one second boundary. Given each FOX will need to transmit for many seconds to provide enough time for a hunter to establish a fix, this one second scheduling granularity will adequate.

The zNEO makes use of a 10mS interrupt to keep the system time updated. The TOY clock is read as startup (triggered by the **TIME** command) and the the system advance time by 100mS at every 10mS interrupt.

8.2 Scheduling Algorithm

Now we are diving into the heart of the scheduling algorithm.

The schedule is described using two numbers, a period and an offset. Since our granularity is one second, these numbers will be expressed in seconds.

As you might expect, if we give 4 FOX Transmitters a period of 60 seconds, they will broadcast once per minute. If we rather casually assume everyone get equal time, that allows 15 seconds per transmitter.

So far, so good, but there must be a means of keeping them from operating at the same time. As mentioned above, assume for the moment that the clocks in each FOX are all synchronized (this should be the case as we updated the time last night!).

8.2.1 Scheduling Period

This is the repeat cycle, in seconds.

Every *N* seconds the cycle repeats (or, you might say, starts).

The start of the period occurs when the system clock divided by the period produces a remainder of zero. This point is the functional basis for the scheduling algorithm. At any time, the software can determine **when** *within the period* by dividing the system time by the period and taking the remainder.

8.2.2 Scheduling Offset

This is the offset into the repeat cycle.

Given that we can calculate the **when** *within the period* number, we can start a transmission when this number matches our scheduling offset.

If the 4 FOX units in our rambling example used different offsets, such as 0, 15, 30, and 45, and the system clocks are all the same, they will transmit their messages in sequence. The messages, if they are less than 15 seconds in length, will not occur at the same time.

You can well imagine things will get garbled a bit should the message length exceed 15 seconds.

8.2.3 Clock Synchronization

Clock synchronization is achieved by slaving the time in the slave transmitters to a host system. This requires that all of the transmitters have their clocks updated prior to the event (like, last night...).

8.3 Scheduling Flexibility

You may notice by now, that a first order schedule might be something like the following:

Table 8: Scheduling Example 1

Unit Serial	**Period**	**Offset**
FOX 1	300	0
FOX 2	300	60
FOX 3	300	120
FOX 4	300	180

This gives a 5 minute cycle with no one transmitting in the 4th minute of the cycle.

We could specify a schedule like this:

Table 9: Scheduling Example 2

Unit Serial	**Period**	**Offset**
FOX 1	100	0
FOX 2	300	50
FOX 3	300	150
FOX 4	300	250

We keep the 5 minute (i.e. 300 seconds) cycle, but have FOX-1 transmitting three times as often as FOX-2, FOX-3 and FOX-4. The modular arithmetic used to calculate when to transmit keeps everything in synchronization providing only the scheduling values and a synchronized clock.

8.4 TOY Clock

The hardware includes a battery backed *Time Of Year* clock. This *TOY* clock keeps a 32 bit counter that increments once per second. This allows a set of fox transmitters to be configured and time locked and then they can be powered up independently of each other.

Typically the TOY clock is read using the **TIME** command when the power is applied (i.e. in the INI= sequence). The system then keeps track using the 10mS interrupt.

We can force reading the TOY clock more often, if needed, by simply issuing the **TIME** command at the end of our **S*-** sequences. This might be used to deal with clock skew caused by high activity levels during message transmission causing lost 10mS interrupts.

8.4.1 Clock Characteristics

The clock itself is a MaximIC DS1672. This is a simple 32 bit battery backed counter that increments once per second.

The time must loaded into the clock using the command subsystem prior to first use. The DS1672 has a backup battery that maintains the time when the system is not powered.

The 32KHZ oscillator will run for a few months without charging the on-board battery. If the unit is not to be used for several weeks, the battery maintenance circuit should keep the backup battery at peak charge as long as the main bayyery is present.

8.5 Code Generator

The code generator is taken almost directly from the the radio audio interface from 2013. All that is changed is the interrupt handling for the zNEO that differs slightly from the eZ8-Encore used in the radio audio interface.

A short message, of up to about 25 characters, is fed into the code subsystem for transmission. Long messages are built of any reasonable number of short buffers.
The test message is translated into code chips that then drives the interrupt routine to control the tone generator.

8.5.1 WPM Rate Control

The speed with which the code message is sent is controlled through the command interface and specified directly in words per minute. The WPM rate is stored in the configuration structure by command decoding. During message transmission, this WPM rate is used to set the rate that the interrupt service routine is activated. The ISR activation rate is programmed to be the length of a single *DIT*.
All subsequent timing is directly linked to the duration for a single *DIT*.

8.5.2 Chipping

This *chipping* refers to the method of controlling the tone generator from the ISR. Each pass through the ISR causes a decision to be made concerning the state of the tone generator either enabled or disabled. Each *chip* has and on/off bit and a count. The on/off control bit is used to drive the enable control on the tone generator and the count is decremented each pass through the ISR until it reaches zero. When this count reaches zero, the chip is discarded, and the ISR moves on to the next chip in the buffer.

Each chip controls a single on/off transition. The letters *O* and *S* both have 6 chips, 3 on periods, 2 off periods and an inter-character or inter-word gap. Although they both require 6 chips to store they take rather different times to execute.

After all the chips in the buffer have been sent, the ISR disables itself and leaves an indication for the main-line code that the buffer has been sent. The main-line code can then fetch the next message fragment and translate from clear-text into a chipping buffer.

8.5.3 Chirping

This *chirping* refers to sending modulated or unmodulated RF in short *chirps.* The **CHRP** command controls this with details of setting this up in section 9.2.23 on page 124

8.5.4 Morse Translation

Translation from clear-text to chips makes use of a lookup table. The lookup table consists of an ASCII character key and a short buffer containing the dots and dashes. The code routine takes the incoming message text, byte by byte, and builds the chipping buffer using the dot and dash indicators in the translation table. Inter character spacing is driven by punctuation; a space or comma generates inter-word timing and a period generates inter-sentence timing.

The code generator timing is controlled by the parameters in the **CWPM** command. The parameters specify the *chipping rate* (expressed in nominal *words per minute*), the *DIT* time (normally 1), the *DAH* time (normally 3), the *INTER CHARACTER* time and the *INTER WORD* time.
The *chipping rate* field is used to calculate the register values that are loaded into the timer that generates the interrupt stream driving the code generator.
The translation process proceeds through the input buffer byte-by-byte performing a lookup in the **CODE TABLE** to find *DIT/DAH/INTER CHARACTER.* These are then loaded into the chipping buffer with the **CHIP COUNT** field coming from the values provided in the **CWPM** command.

Once the entire message has been translated into a chipping buffer, the interrupt for the code generator is enabled and the interrupt handler proceeds through the chipping buffer, turning the audio source on/off as specified.

Listing 2: cmd_code.c-122

```
                {' ',  R"W"        },   // inter-word spacing
                {'_',  R"W"        },   // inter-word spacing
                {',',  R"W"        },   // inter-word spacing
                {'.',  R"S"        },   // inter-sentence spacing
                {':',  R"S"        },   // colon (in time string)
                {CODE_DIT, R"."    },    // dit
                {CODE_DAH, R"-"    },    // dah
```

Timing characters.

Listing 3: cmd_code.h-59

```
#define CODE_DIT           '<'
#define CODE_DAH           '>'
```

Characters definitions for **CODE_DIT** and **CODE_DAH**.

Listing 4: cmd_code.c-131

```
                        {'!',  R".-..-." },   // bang
                        {'/',  R"-..-."  },   // slash
                        {'&',  R".-..."  },   // ampersand
                        {'=',  R"-...-"  },   // BT (begin 2 lines)
                        {'+',  R".-.-."  },   // AR (all received)
                        {'-',  R"-....-" },   //
                        {'$',  R"...-..-"},   //
4735                    {'@',  R".--.-." },   //
                        {'&',  R".-...." },   // ampersand (wait)
                        {';',  R"-.-.-." },   //
                        {'(',  R"-.--."  },   //
                        {')',  R"-.--.-" },   //
                        {'\'', R".----." },   //
                        {'_',  R"..--.-" },   //
```

Punctuation characters.

Listing 5: cmd_code.c-146

```
{'A', R".-    "  },
{'B', R"-...  "  },
{'C', R"-.-.  "  },
{'D', R"-..   "  },
{'E', R".     "  },
{'F', R"..-.  "  },
{'G', R"--.   "  },
{'H', R"....  "  },
{'I', R"..    "  },
{'J', R".---  "  },
{'K', R"-.-   "  },
{'L', R".-..  "  },
{'M', R"--    "  },
{'N', R"-.    "  },
{'O', R"---   "  },
{'P', R".--.  "  },
{'Q', R"--.-  "  },
{'R', R".-.   "  },
{'S', R"...   "  },
{'T', R"-     "  },
{'U', R"..-   "  },
{'V', R"...-  "  },
{'W', R".--   "  },
{'X', R"-..-  "  },
{'Y', R"-.--  "  },
{'Z', R"--..  "  },
```

Alpha characters.

Listing 6: cmd_code.c-173

```
{'0', R"----- "  },
{'1', R".---- "  },
{'2', R"..--- "  },
{'3', R"...-- "  },
{'4', R"....- "  },
{'5', R"..... "  },
{'6', R"-.... "  },
{'7', R"--... "  },
{'8', R"---.. "  },
{'9', R"----. "  },
{0,NULL}                          // MUST be the last entry!
};
```

Number characters.

Listing 7: cmd_code.h-14

```
//      MAX_CHIPS is the maximum number of keying command we will handle...
#define MAX_CHIPS        300
//
//      CHIP
//
//              +——+——+——+——+——+——+——+——+
//              |     |     |     |     |
//              +——+——+——+——+——+——+——+——+
//                ^   ^   ^  \              /
//                |   |   |   \            /
// CHIP_ACTIVE ——+    |   |    \          /
//                    |   |     \        /
// CHIP_KEY_ON ————————+  |      \      /
//                        |       \    /
// CHIP_GAPPED ——————————+         \  /
//                                  |
// CHIP_COUNT (i.e. interrupt count)——+
```

Chipping Buffer Layout.

This is the buffer used at interrupt level to provide on/off control of the CW tone. The chipping rate (i.e. the WPM rate) is driven by the interrupt arrival rate. The interrupt timer is set to the time of a single chip (i.e. dit) .

Each element in a character takes two bytes in the buffer. First byte has the audio *ON* time and the second has the audio *OFF* time.

Longer inter-element spacing is encoded with a larger value in the **CHIP_COUNT** field.

8.5.5 Interrupt Initiation

The interrupt initiation entry point calculates the values that are to be loaded into the timer control registers to run the ISR at the target rate. The buffer address, passed from the calling routine, is stored where the ISR can access it and the interrupt for the timer is enabled and the timer itself is enabled.

The interrupt initialization routine then waits for the message to be sent before returning control to the main-line code.

8.5.6 Timeout Conditions

As the activities described above are well bounded, we do not expect to require a watch-dog facility. There is none-the-less a timer that is started at the beginning of each message buffer that will release the wait if the message takes too long to send.
Should this occur, is is an indication that the specified WPM rate is too slow of the message itself is too long.
This is remedied by changing the offending schedule.

8.6 Audio

Audio generation is program controlled, minimal hardware assists are involved in this process. The zNEO is operating a busy-wait loop to move data from FLASH (or FRAM) to the PWM block in the zNEO.

Errors in the directory records can cause problems.

8.6.1 Directory Record

One record describes each audio clip. If the audio clip is an 8-bit mono RIFF/WAVE file, the second form should be used with the sample rate and sample count coming directly from the RIFF/WAVE header.

TALK=<*name*>,<textitstart>,<textitcount>,<textitaddress>

TALK=<*name*>,<textitstart>

8.6.2 Waveform Data

Loaded using *Intel Hex* records.

This is raw 8-bit PCM data. It may have a RIFF/WAVE header that describes the length of the data block and the sample rate. Sample rates are limited to 4K/s, 5K/s, 10K/s, or 16K/s. Date width must be 8 bits and the data must be single channel.

The data will not be used if the RIFF/WAVE header indicates stereo of 16 bit data.

8.7 Status Reports (commanding)

Whenever an activity occurs within the system, a status report is produced on the control port (USB or the 3.5mm serial jack).
These reports have a style to them that is intended to make decoding by a host computer easy to implement.

All status messages consist of a 3 letter key, a numeric command index followed by a comma, a numeric status value that is the followed by an asterisk.
Additional text may follow the asterisk that is intended to be human readable.

These reports tend to be rather chatty, with a great deal of information used to debug the software. There is also extensive help text that can be called up when needed. The time required to pass all this traffic seems like it would get in the way when operating, but the volume of text traffic resulting from the commands that are used to actually implement a fox sequence is not that large.

8.7.1 RDY

When the input processing loop becomes ready to accept a command, a RDY report is sent out on the control port. A command may come in through the serial channel. The system may also process an internal sequence if scheduling is enabled.

The numeric command index should be zero.

The numeric status value indicates the current state of the *run flag*. The *run flag* must be set to 1 in order for scheduled sequences to execute.

> Example RDY reports:
>
> ```
> RDY00,00* (Sp=0xBF94)+1870 00:00:04.790
> RDY00,01* (Sp=0xBFD8)+1938 15:01:13.970
> ```
>
> The report contains some diagnostics and the system time.
>
> Note that the first example, the second status field is zero, indicating that the run flag is currently cleared. No scheduling occurs when the run flag is cleared.
>
> In the second example, the second status field is one, indicating that the run flag is currently set. The Fox Transmitter will transmit traffic when the scheduling point is reached.
>
> The diagnostic part of the line is the current stack pointer location (hexadecimal) and the free space (decimal) on the stack. The zNEO has only 4KB of RAM, so the would expect the free space on the stack to report around 1800 bytes.
>
> The system time is taken from the time field that the system keeps. It requires setting from the TOY clock to have any relation to the real world.

Sending a *carriage return* should cause a RDY report to be returned. Use this behavior to check the clock alignment (i.e. how well it matches wall-clock time).
An empty command (*carriage return* alone) will clear the run flag.

8.7.2 STS

When a command is completed, this status message reports on the success of the command. This report, where the **STS** is uppercase, only occurs when the system is ready to accept command traffic from the serial port. In other words, a running sequence never sends out **STS** is uppercase.

The numeric command index is the internal index of the command. This numeric value is generated by the first step of the command decoding process. It will be positive if the 4 letter command stem is recognized. Unrecognized commands will have a negative value in this first field.

The numeric status value indicates if the arguments to the command are correctly formed and have been accepted. A positive value indicates the command was correctly formed and has been executed. A negative value here indicates that the arguments were mal-formed. *run flag*. The *run flag* must be set to 1 in order for scheduled sequences to execute.

An example STS report:

```
STS01,47* Handler_HELP (cmd_help.c*)  2.80 Sec
```

The report shows the handler that produced the text along with the module in which the handler *lives*.

Also note the execution time of the command is shown.

8.7.3 sts

This report (note that *sts* is lowercase) is part of any intermediate results.

The numeric command index is the internal index of the command.

The numeric status is normally a simple counter that tallies each of the lines generated by the command.

An example sts report:

```
sts01,01*   1 HELP SYS  Help Menu and Items
sts01,02*   2 HELP SYS  <string>   matching help items
sts01,03*   3 ONCE SYS  <name>     Test run the named sequence
```

The report content is unique to each command and presents status and diagnostic information about the execution of the command.

The sts01,01* portion of the report is always formatted the same. The numeric command index followed by a sequence number. The asterisk marks the end of the report header.

8.8 Signon Report

This report is produced when the system is reset. This occurs whenever the reset pin on the zNEO is cycled *low to high*.

Sample signon message:

```
sts01,00* ****************************************************************
sts01,01* KC0JFQ  FOX Transmitter  V3.27
sts01,02*   Z16F2810AG20EG
sts01,03* Tools Ver: 20230510  <ROM:25664 EROM:86622 Flash:112286>
sts01,04* Flash Prog 04.25.03 MB85RS256TY    FLASH_FRAM       Fujitsu   256K-bits   1024-records CMD3
sts01,05* Flash WAVE 9D.7E.FF    25LD040    FLASH_PAGE          ISSI  4096K-bits     62-seconds CMD4
sts01,06* ****************************************************************
```

8.8.1 sts01,01: Version

Author, Project Name, and Build Version from the *MAKEFILE*.

8.8.2 sts01,02: zNEO Hardware

zNEO part number from the zNEO device.

The older boards use an 80 pin flat pack while the newest board make us of a 64 pin flat pack.

Both packages are available as a Z16F2810 or Z16F2811. These are both 128KB flash devices with an identical peripheral complement. The Z16F64 and Z16F32 devices have a smaller program flash that is too small to hold the current software.

8.8.3 sts01,03: Tools Ver:

This line displays the date string from the ZiLOG development tools used to build the software system for the fox transmitter. It is in the form of year/month/day to reflect when the tools were compiled at the vendor.

Also appearing on this line are the flash memory allocations in the zNEO processor. The compiler/linker from ZiLOG breaks the zNEO flash allocation into two groups called *ROM* and *EROM*. The memory use under each location counter is reported. The total memory use is also reported (it is the sum of *ROM* and *EROM* allocations).

8.8.4 sts01,04: Flash Prog (U3)

JEDEC ID from the FRAM device.

If the device appears in the *device table* (in the fox transmitter software) the part number and capacity are reported.

8.8.5 sts01,05: Flash WAVE (U12)

JEDEC ID from the FLASH device.

If the device appears in the *device table* (in the fox transmitter software) the part number and capacity are reported.

8.9 Status Report (STAT I command)

This command is used to dump the module identification strings.

This will emit a number of lines, each with the ID string from one of the modules that make up the operating software of the fox transmitter system.

The string has the compile time of the individual module, the version number of the module, and the module name.

Sample STAT I report (truncated):

```
sts09,00* <<<---  STAT  ********  STAT  --->>>
sts09,01* IDENT:Mar  4 2024 10:54:51 V3.20 fox_73181.c*
sts09,02* IDENT:Mar  4 2024 10:54:59 V3.03 gpio_local.c*
sts09,03* IDENT:Mar  4 2024 10:54:58 V2.03 timer_local.c*
sts09,04* IDENT:Mar  4 2024 10:54:59 V1.00 uart_local.c*
sts09,05* IDENT:Mar  5 2024 16:15:39 V3.05 flash_local.c*
sts09,06* IDENT:Mar  4 2024 10:54:59 V1.03 i2c_local.c*
sts09,07* IDENT:Mar  4 2024 10:54:59 V1.00 analog_local.c*
sts09,08* IDENT:Mar  4 2024 10:54:51 V3.07 command.c*
sts09,09* IDENT:Mar  4 2024 10:54:52 V1.01 cmd_help.c*
sts09,10* IDENT:Mar  4 2024 10:54:51 V1.02 cmd_code.c*
sts09,11* IDENT:Mar  5 2024 08:40:06 V3.17 cmd_stat.c*
sts09,12* IDENT:Mar  5 2024 08:31:24 V1.06 cmd_conf.c*
sts09,13* IDENT:Mar  4 2024 10:54:54 V0.10 cmd_message.c*
sts09,14* IDENT:Mar  4 2024 10:54:54 V3.10 cmd_sa818b.c*
sts09,15* IDENT:Mar  4 2024 10:54:54 V1.05 cmd_si5351.c*
sts09,16* IDENT:Mar  4 2024 11:00:23 V1.01 cmd_ics525.c*
sts09,17* IDENT:Mar  4 2024 10:54:56 V1.01 cmd_sys.c*
sts09,18* IDENT:Mar  4 2024 10:54:56 V1.00 cmd_proc.c*
sts09,19* IDENT:Mar  4 2024 10:54:56 V1.00 cmd_sched.c*
sts09,20* IDENT:Mar  4 2024 10:54:56 V1.01 cmd_time.c*
sts09,21* IDENT:Mar  4 2024 10:54:56 V3.04 cmd_test.c*
sts09,22* IDENT:Mar  4 2024 10:54:56 V3.05 cmd_voice.c*
sts09,23* IDENT:Mar  4 2024 10:54:57 V1.03 cmd_battery.c*
sts09,24* IDENT:Mar  4 2024 10:54:57 V3.02 cmd_flash.c*
sts09,25* IDENT:Mar  4 2024 10:54:57 V3.05 cmd_frequency.c*
sts09,26* IDENT:Mar  4 2024 10:54:57 V1.01 cmd_wav.c*
sts09,27* IDENT:Mar  4 2024 10:54:58 V1.02 fox_schedule.c*
sts09,28* IDENT:Mar  4 2024 10:54:51 V2.00 radio_control.c*
sts09,29* IDENT:Mar  4 2024 10:55:00 V1.00 daytime.c*
sts09,30* IDENT:Mar  4 2024 10:54:51 V1.00 modulus.c*
sts09,31* IDENT:Mar  4 2024 10:55:00 V3.13 intel_hex.c*
sts09,32* IDENT:Mar  4 2024 10:54:57 V1.00 test_help.c*
sts09,33* IDENT:Mar  4 2024 10:54:57 V1.00 test_spi.c*
sts09,34* IDENT:Mar  4 2024 10:54:57 V1.00 test_zneo.c*
sts09,35* IDENT:Mar  4 2024 10:54:58 V1.00 test_i2c.c*
sts09,36* IDENT:Mar  4 2024 10:54:58 V1.02 test_gpio.c*
sts09,37* IDENT:Mar  4 2024 10:54:58 V1.00 test_batt.c*
sts09,38* IDENT:Mar  4 2024 10:54:58 V1.01 test_code.c*
sts09,39* IDENT:Mar  4 2024 10:54:58 V1.01 test_serial.c*
sts09,40* IDENT:Mar  4 2024 10:54:58 V1.00 test_config.c*
sts09,41* IDENT:Mar  4 2024 10:54:51 V1.00 ident_scan.c*
sts09,42* KC0JFQ  FOX Transmitter
```

The rest of the STAT report has been removed, leaving only the module report detail lines.

8.10 Status Report (STAT command)

This command is used to dump the system status, reporting on all of the programmable system settings. This also includes hardware state and settings that are not programmable, but may change as determined by installed hardware or readings from the A/D subsystem, etc.

Sample status report:

```
sts09,00* <<<---  STAT  ********  STAT  --->>>
sts09,01* KC0JFQ  FOX Transmitter
sts09,02* Software Bld:   Mar  5 2024  08:40:06
sts09,03* System Time:    15:26:01.800 (73561)
sts09,04* Epoch Offset:   19:00:00     (68400)
sts09,05* TOY Clock:      000BAB5A 00 00; Osc ON, Charge Disabled
sts09,06* Sys Upd Flg:    Not Set
sts09,07* Conf Jumpers:   NOT_Master Test
sts09,08* Flash Prog  U3: 04.25.03 MB85RS256TY    FLASH_FRAM       Fujitsu   256K-bits   1024-records CMD3
sts09,09* Flash WAVE U12: 9D.7E.FF    25LD040    FLASH_PAGE          ISSI  4096K-bits     62-seconds CMD4
sts09,10* Flash HEX Dev:  WAVE
sts09,11* Battery, Idle:  9.404V(0x03C3)[9.766e-03]  22mA(0x002E)
sts09,12* Analog Others:  Reg-5V: 5.098V(0x020A)[9.766e-03]  Switch: 0.000V  CdS-Cell: 0.000V
sts09,13* UART buffer:    143 (NET:0, USB:71)
sts09,14* <<<--- Scheduling  PARAMETERS --->>>
sts09,15* <<<--- TRANSMITTER PARAMETERS --->>>
sts09,16* Callsign:       KC0JFQ
sts09,17* Nickname:       FOX21
sts09,18* zNEO Port Bits: OUT:E0,01,05,20,00,00,00,00 IN:F0,01,35,10,00,80,00,04
sts09,19* Radio Config:   0x1F01D5 DRA818 State-T0 CTL TX_ENA TONE PWMHO 5MON SWIT PHOTO IMON BMON
sts09,20* Frequency:      144.150 (Xtal: 0.000 MHz)
sts09,21* CW config:      30 WPM 1,3,7,14 [0x186A] 0.600KHz
sts09,22* State Delays:   T0:100 T1:2000 T2:50 T4:50 T5:10
STS09,23* Handler_STAT I (cmd_stat.c*)  0.90 Sec
```

8.10.1 Software Bld:

Date and time the software was compiled. This date-stamp is taken from the *cmd_stat.c* module.

8.10.2 System Time:

The time currently stored (and updated) by the fox transmitter software.

This time is nominally stored as UNIX time (in terms of UT).

8.10.3 Epoch Offset:

Time zone offset from UT (hours). West is negative. Iowa is -5 in the summer.

8.10.4 TOY Clock:

Register dump of the DS1672 TOY clock.

32 bits of the time register. This is arranged with the MSB on the left of the field.

8 bits of register 4 where the **ESOC** bit appears in bit D7.

8 bits of register 5 where the charge control bit appear.

8.10.5 Sys Upd Flg:

System Update Flag indicates that the system time field has been loaded from the TOY clock and when the SI5351 has been initialized.

8.10.6 Conf Jumpers:

Shows the presence of the **TEST** and **MAS** jumpers.

8.10.7 Flash Prog U3:

Dump of the JEDEC ID bits from the U3 position.

This is where a FRAM device should be installed.

Some devices do not support the *Read-ID* command leaving this field set to all 1s. If this is the case or when no device is installed, the device is treated as a 64Kb device. If installed device is smaller, loading commands into the device may cause a wrap-around to occur and overwrite commands at the beginning of the device (**don't do that!**).

When no device is installed, reading will yield all 1s which appears as an erased device.

8.10.8 Flash WAVE U12:

Dump of the JEDEC ID bits from the U12 position.

This is where a FRAM device should be installed.

Some devices do not support the *Read-ID* command leaving this field set to all 1s. If this is the case or when no device is installed, the device is treated as a 64Kb device. This is too small for practical use and should indicate a bad device or no device installed.

8.10.9 Flash HEX Dev:

This is the device on which the Intel HEX commands operate.

This is for the 102-73161 boards where a single device appears on the circuit board.

For the 102-73181 boards this should report **WAVE** indicating the Intel HEX commands work on U12.

8.10.10 Battery, Idle:

Voltage and current readings when the system is idle.

The 102-73161 boards will not report current.

8.10.11 Battery, TX:

Voltage and current readings when the system is transmitting.

The 102-73161 boards will not report current.

8.10.12 UART buffer:

Size of the UART buffer and the maximum number of bytes that have been used (i.e. characters have been stored).

This is a sanity check for software debugging.

8.10.13 Callsign:

This reports the callsign stored by the **CALL** command.

Prior to the occurrence of a **CALL** command, this field contains *SOS SOS SOS* to indicate that a problem exists.

8.10.14 Nickname:

This reports the callsign stored by the **NAME** command.

Prior to the occurrence of a **NAME** command, this field is empty. Field substitution results in nothing being stored.

8.10.15 zNEO Port Bits:

Diagnostic dump of the zNEO general IO port bit patterns.

8.10.16 Radio Config:

Dump of the radio configuration bits.

The **CONF** command is used to set/clear these bits and this is a quick report of their status.

The **CONF** command with no arguments provides a detailed report of the current bit states along with the mnemonics used to set/clear them.

8.10.17 Frequency:

Currently selected transmit frequency.

8.10.18 CW config:

Code chipping parameters.

The word rate and timing parameters for the code generator.

8.10.19 State Delays:

To accommodate the various RF generator configurations, **BEGN** and **DONE** commands implement the delays defined in figure **??** on page 13.

The *T1* state is set to milliseconds for the ICS307, ICS525, and SI5351, while the DRA818/SA818 modules require about 2 seconds to awake from a powered down state.

8.10.20

...

8.11 I2C

Notes on the operation of the I2C subsystem.

The current software is a bit-banged implementation of the I2C protocol. The volume of traffic is so low there is not driving need to use the I2C hardware in the4 zNEO.

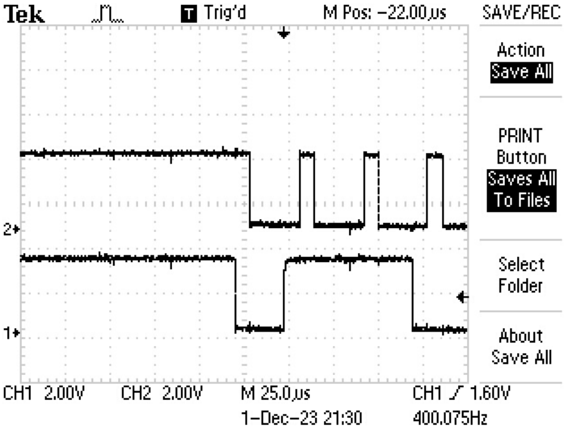
The inter-bit timing can be seen on the following 'scope plots.

Nominally, the SDA line is steady when the clock line is high. The I2C implementation keeps the SDA line steady from at least one µS prior to a rising clock edge to at least one µS after a falling edge.

The START and STOP conditions expressly violates the rule.

SCK is shown on the top (probe 2). SDA is shown on the bottom (probe 1).

8.11.1 I2C START



5456

Figure 20: I2C START

Here the SDA line (bottom line, channel 1) goes low when the clock line (top line, channel 2), SCK, is high.
This violates the *rules* and is used to indicate a START condition.

The controller (in our case the zNEO) follows the START condition with a 7 bit device address and a 1 bit direction flag.

8.11.2 I2C Data

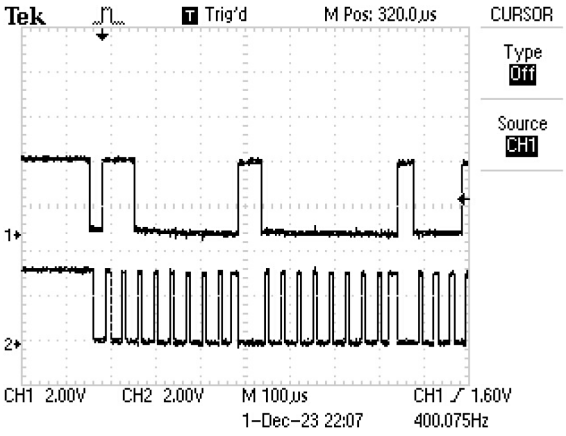


Figure 21: I2C Data

Here we see the I2C device address byte, a data byte, and the start of a second data byte.

Looking at the last bit of the address byte, we see a **zero** indicating this is a write transaction. The controller continues to drive the SDA line as it shifts out successive 8 bit words.

Take note of 8 data bits (the narrow clock pulses) and one acknowledge bit (the slightly wider clock pulse before the small gap. The target device drives the SDA line low after is has completed processing the last data bit. In this example, the target device begins driving SDA low before the controller stops driving, so we do not see indication that the target device can't keep up.

8.11.3 I2C STOP

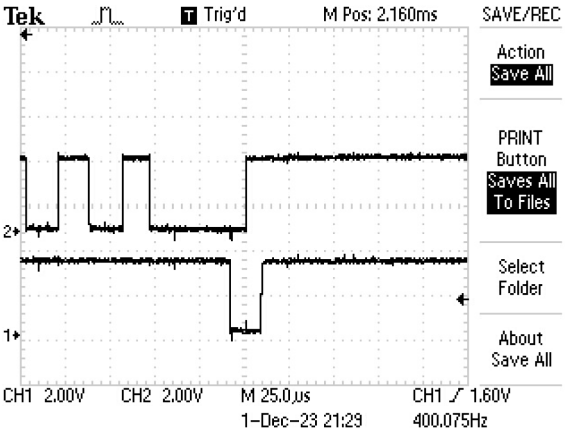


Figure 22: I2C STOP

The last event of the data transaction is the controller asserting the STOP condition.

As with the START condition, a transition on the SDA line occurs when the SCK line is high, *violating* the rules. The target recognizes this as the STOP condition and stops sending data during a read or stops looking for data during a write.

8.11.4 I2C Write Buffer

The write transaction proceed as shown in the above example.
The controller asserts the START condition, sends out the 7 bit target address followed by a **write** bit (a zero bit).
The next byte has the 8 bit device register address.
This is followed by one or more data patterns that are loaded into successive registers in the target device.

8.11.5 I2C Read Buffer

A read transaction involves a write of the device register address follows by a read of the device data.

A write transaction with a device address and a device register address is sent. This is just 2 I2C bytes followed by a STOP condition.

The controller then starts a new transaction. The controller asserts the START condition, sends out the 7 bit target address followed by a **read** bit (a one bit).

The controller leaves the SDA line inactive, either driving it to a one (in the case of open-drain pin) or tri-stating the pin. The target updates the SDA line as the controller pulses the clock line.
The controller continues to send 8+1 pulses for each needed input byte. When the driver (in the controller) has satisfied the input request it asserted the STOP condition to end the transaction.

8.12 SPI

Access to SPI peripherals, the two memory devices, uses the ESPI function of the zNEO. The chip select lines, one for each memory device, are managed as GPIO output lines.

Buffer space in the zNEO is limited, so the nominal record size for the memory devices is 32 bytes.

Audo data, from the waveform memory device (U12) is managed a byte-at-a-time. The serial clock rate is programmed to be 8 time the sample rate so that data bytes are read from the SPI memory device at the target sample rate. No additional clock resources are needed to deal with flowing audio data.

8.13 Message Processing

A *message*, as sent by a properly configured and sequenced fox transmitter, consists of a *header*, *message body*, and *trailer*. These must be properly loaded into the FRAM in order to have the system generate reliable and *rules compliant* message traffic.

The *message body* may consist of as many message fragments as desired. The message fragments are sent using the CODE, TALK, SCAL, and BATx commands. CODE generates Morse traffic from the text in the command. TALK generates audio from the audio file specified in the command. SCAL sends the system callsign which was stored using the CALL command as part of the setup commands. Finally, the BATx commands generate battery status messages to allow convenient monitoring of system health (i.e. how much operating time is left given the reported battery voltage).

The *header*, initiated using the BEGN command, configures the RF subsystem for operation by switching on power and programming the frequency synthesizer and then sending a signon message with the station callsign.

The *trailer*, initiated using the DONE command, send the station callsign, here to comply with the rules, and returns the RF subsystem to a low power state.

8.14 Frequency Selection

Frequency selection is achieved by multiple means. The software is built with small internal tables containing bit patterns that are used to configure the clock synthesizer.

For the DRA818/SA818 the frequency string may be used directly to setup the transceiver module.

A third method is to store the register setup patterns in FRAM using the **ESAV=** command.

8.14.1 Frequency Tables in FRAM

Frequency setup tables can be stored in program FRAM. This avoids the need to have a ZiLog programmer to update the flash memory in the zNEO device to use a new frequency.

Tables for the SI5351 can more easily be tailored to address issues with the reference clock. In all of the prototype SI5351 builds, the 20MHz reference crystal was operating off frequency and required trimming the SI5351 register values. If you have a spectrum analyzer available to find the carrier frequency, the SI5351 frequency calculation utility can be adjusted to eliminate frequency errors.
A sample frequency table for storage in FRAM as generated by the *si5351a_calc* utility:

```
REM-  Call Line: "/home/wtr/Fox_Tx_73181/trunk/si5351a_calc
                          -T2
                          -F 144.250,144.300,5
                          -o-14
                          -e freq_5351.fox "
REM- Output File: freq_5351.fox
REM- SI5351 Xtal:  20.000MHz
REM- Freq Offset: -14.000KHz
esav 144.250=13A2,A1B80,F4240
esav 144.255=13A2,D0980,F4240
esav 144.260=13A3,0B540,F4240
esav 144.265=13A3,3A340,F4240
esav 144.270=13A3,6913F,F4240
esav 144.275=13A3,97F40,F4240
esav 144.280=13A3,C6D40,F4240
esav 144.285=13A4,01900,F4240
esav 144.290=13A4,30700,F4240
esav 144.295=13A4,5F500,F4240
```

The *esav* lines store the register patterns for the SI5351. This would nominally be a set of frequencies that are not present in tables programmed into the program flash in the zNEO.

The **FREQ** command uses the frequency string passed in the command to look for a matching entry in the FRAM. If a matching entry is found in FRAM, the values in the FRAM table are used to configure the SI5351. Conflicts are resolved by using the values from the FRAM tables The FRAM can, therefore, be used to override the internal table.

We can inspect the entries in the FRAM using the **EDMP=** command:

```
        edmp 145.
        sts35,00* Handler_EDMP (cmd_flash.c*)  (  13) 145.000=13BF,70E40,F4240
        sts35,01* Handler_EDMP (cmd_flash.c*)  (  14) 145.025=13C0,671FF,F4240
        sts35,02* Handler_EDMP (cmd_flash.c*)  (  15) 145.050=13C1,5D5C0,F4240
        sts35,03* Handler_EDMP (cmd_flash.c*)  (  16) 145.075=13C2,5397F,F4240
        sts35,04* Handler_EDMP (cmd_flash.c*)  (  17) 145.100=13C3,49D3F,F4240
        sts35,05* Handler_EDMP (cmd_flash.c*)  (  18) 145.125=13C4,40100,F4240
        sts35,06* Handler_EDMP (cmd_flash.c*)  (  19) 145.150=13C5,364C0,F4240
        sts35,07* Handler_EDMP (cmd_flash.c*)  (  20) 145.175=13C6,2C87F,F4240
        sts35,08* Handler_EDMP (cmd_flash.c*)  (  21) 145.200=13C7,22C3F,F4240
        sts35,09* Handler_EDMP (cmd_flash.c*)  (  22) 145.225=13C8,18FFF,F4240
        sts35,10* Handler_EDMP (cmd_flash.c*)  (  23) 145.250=13C9,0F3BF,F4240
        sts35,11* Handler_EDMP (cmd_flash.c*)  (  24) 145.275=13CA,0577F,F4240
        sts35,12* Handler_EDMP (cmd_flash.c*)  (  25) 145.300=13CA,EFD80,F4240
        sts35,13* Handler_EDMP (cmd_flash.c*)  (  26) 145.325=13CB,E613F,F4240
        sts35,14* Handler_EDMP (cmd_flash.c*)  (  27) 145.350=13CC,DC4FF,F4240
        sts35,15* Handler_EDMP (cmd_flash.c*)  (  28) 145.375=13CD,D28BF,F4240
        STS35,16* Handler_EDMP (cmd_flash.c*)    0.80 Sec
        RDY00,00* (Sp=0xBF94)+1886 14:15:06.110
```

The new frequency entries are not position critical as they are used (i.e. search for) when the **FREQ** command occurs.

9 Commanding

The primary motivation for using the zNEO is to provide enough program memory and raw speed to be able to easily implement the instrument timing methods and to provide a convenient commanding interface to configure the FOX.

On the 102-73181 boards a single port is available to load sequence memory (FRAM) and audio (FLASH) memory into the FOX. Commanding through the control port is identical to previous units. The time network port has been eliminated.

Commands consist of a *command keyword* that is delimited by a comma or space. The delimiter is restricted to these character to allow the system to differentiate between commands and files. Filenames are delimited (or followed) by an equal (=) character.

The need for this requirement becomes evident when dealing with the TALK command where the **TALK** keyword performs two different roles. One being the command to send an audio clip (**TALK** *filename*) and the other being a directory entry (**TALK=** *name, start, size, rate*).

9.1 Command Status

The command completion status reports are somewhat formalized and intended to allow for host controlled operation.

Remote host control makes little sense for fox hunts, but it does provide for a means of managing the loading of command sequences and loading of Intel HEX files.

All traffic to the host system through the serial port is formatted as a report beginning with a 3-character key, a **numeric index**, and a **reporting value**. This is then delimited with an asterisk. Text after the asterisk is free-form and varies with the individual commands.

The numeric index is simply the index into the command dispatch table. Although the specific numbers are nominally static, they can change when commands are added or removed.

For what-its-worth, the **HELP** command shows the index number associated with each command.

9.1.1 sts reports

Any message traffic that is generated by a command is prefixed by a **sts** key to indicate this is status reporting traffic.

The **HELP** command, for example, reports on all the commands that appear in the command dispatch table, each one starting with **sts** at the very beginning of the line.

Some commands only generate a single status message, in which case no **sts** message appears.

9.1.2 STS report

All commands generate at least one status report. The last status report that is generated has the **sts** key converted to uppercase, so the key becomes **STS**.

This occurs to provide an indication to the host that the command has finished executing and the status code appears in the **reporting value** field.

5821 A negative value in the **reporting value** field indicates a problem was encountered when processing the command.

The status report has appended to the end, an indication of the time the command required to execute.

9.1.3 RDY report

The **RDY00,00*** report is used to indicate to the host that the system is ready to process the next command. The **numeric index** and **reporting value** fields always occur as shown, they are bot set to ZERO.

5837 This message is generated in one place in the code, so the format is somewhat static. The current zNEO stack pointer, the space remaining on the stack, and the current system time all are reported on this line.

The zNEO has a total RAM complement of 4096 bytes.

9.2 Command List

5842

Table 10: Command List 1

Mnemonic	**Class**	**args**	**Description**
HELP	SYS		Help Menu and Items
HELP	SYS	string	matching help items
ONCE	SYS	Sn=	run sequence once
REM-	SYS		Remark, (side-effect: stops schedules)
RUN0	SYS		RUN ALL Schedules
RUN0	SYS	name	RUN Specific Schedule
STAR	SYS	time	Allow schedules to run after specified time
IDLE	SYS		STOP ALL Schedules
STAT	SYS	flag	System Status, (I)ident scan
CONF	SYS	keywords	Hardware Configuration
TOYC	SYS	res (250 2K 4K NONE)	Hi chg rte DS1672 bat
TIME	SYS	time value	Set Time (set DS1672)
D525	SYS	ICS525 control	ICS525 utility

System Control Commands.

Table 11: Command List 2

Mnemonic	**Class**	**args**	**Description**
TIME	SETUP		Time from DS1672 to System (NO Argument!)
EPOC	SETUP	hours	Epoch offset (i.e. time zone)
CALL	SETUP	call	FCC Assigned Callsign
NAME	SETUP	nick	Local Nickname
NICK	SETUP	nick	alias for "NAME", but don’t use it!

System Setup Commands.

Configuration parameters that identify the transmitter and get it in time lock with everyone else.

Table 12: Command List 3

Mnemonic	**Class**	**args**	**Description**
TONE	PGM	freq	Audio Tone (in KHz)
CWPM	PGM	wpm gap1 gap2 gap3	CW Chipping Rate
FREQ	PGM	freq	Frequency (in MHz)
5351	PGM	key value value ...	SI5351 setup group
BEGN	PGM		Key TX and Send Callsign (CW)
CODE	PGM	message	Send Message (CW) up to 22 char
TALK	PGM	file-name	Play Voiced Message (EDMP TALK)
WAIT	PGM	delay	Wait for *delay* seconds
CHRP	PGM	tone per dur cnt	Send chirp train
DONE	PGM		Send Callsign (CW), SK (CW), and unkey TX
BATC	PGM	key setpoint	Transmit CW Battery Report "A" analog read after transmitter enable, "B" analog read before transmitter enable, "E" encode using T and E "V" battery voltage, "I" battery current, "R" 5V rail
BATV	PGM	key	Transmit Vocal Battery Report "A" analog read after transmitter enable, "B" analog read before transmitter enable, "V" battery voltage, "I" battery current, "R" 5V rail

Sequence Commands.

These are the commands that appear in the operating sequences. These define the operating personality of this transmitter.

Table 13: Command List 4

Mnemonic	**Class**	**args**	**Description**
MODS	SCHED	Sname period offset	Modulus Schedule Set
MODC	SCHED	Sname=	Modulus Schedule Clear

Scheduling Commands.

These command load the operating schedule into memory from FRAM using the INI= file.

Table 14: Command List 5

Mnemonic	**Class**	**args**	**Description**
TALK	DIRECTORY	esav TALK= name,Strt,Len,rate	Waveform Directory Entry (appears in FRAM as the TALK= file)

Speech Control.

On the 102-73181 boards there is a second memory device that holds waveform data. This pseudo-command is used to store directory information in the FRAM device to allow access to the waveforms stored in the FLASH device.

Table 15: Command List 6

Mnemonic	**Class**	**args**	**Description**
ESAV	FRAM	NAM=text	Save named record in next free location
EDMP	FRAM	"match string"	Dump active records
EDID	FRAM		JEDEC-ID table dump (PROG & WAVE)
ERAS	FRAM	number	Rewrite record to REM-
EZER	FRAM	number or "DEV"	Erase record to ZERO
ETAB	Diag		Dump the FRAM/FLASH device table

FRAM Control Commands.

These commands are used to access and manage the FRAM device.

Table 16: Command List 7

Mnemonic	**Class**	**args**	**Description**
HERA	FLASH	size start DEV	Hex erase (WAVE device) DEV performs device erase
HDMP	FLASH	length hex-start *	Hex dump (WAVE device)
H56K	FLASH		change bit rate to 57,600 b/S
:hex	FLASH-HEX	:llaaaattddddddddcc	Intel HEX loader (WAVE device)

FLASH Control Commands.

These commands are used to load waveforms into the FLASH device.

Table 17: Command List 8

Mnemonic	**Class**	**args**	**Description**
HALT	TEST		Halt Processor
STOP	TEST		Stop Processor
REST	TEST		Reset System
TEST	TEST		Hardware Test Subsystem

zNEO Control Commands.

These commands are used diagnose the operation of the zNEO processor.

9.2.1 HELP

Table 18: Help Sybsystem

Command sample	**Description**
HELP *string*	Display commands that match string
HELP	Display full list of commands

Action:

Display the list of available commands.

Help text is returned to the serial port.
Providing an argument will perform a substring match such that only the text that matches the provided string is displayed.

Arguments:

Match *string*.

Returns:

STSxx,xx* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

9.2.2 ONCE

Table 19: Run a sequence ONCE

Command sample	**Description**
ONCE *Sn=*	Run sequence **n** one time

Action:

Run the specified sequence one time.

Arguments:

Sequence Name *Sn=*.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Note that the full filename of the sequence must be given in the command or the lookup will fail.

This command is provided to allow the execution time of a sequence to be measured. The final status report (*sts00,11* Execution Time: nn.nn0)* may be used as the message execution time as
6183 it is measured from when the *ONCE* command is recognized to where control is returned to the *ONCE* command manager.

9.2.3 REM-

Table 20: Remark

Command sample	Description
REM-	Remark (no operation)

Action:

This is not acted upon.

May be used to embed remarks or comments in the FRAM.

6229 This command is used to remove a record from FRAM when using the ERAS (section
9.2.33) command. This rewrite avoids altering the end of the FRAM file system

Arguments:

None.

Returns:

STSxx,xx* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

9.2.4 RUN0

Scheduling Control.

Table 21: Scheduling Control

Command sample	**Description**
RUN0 *schedule*	run selected schedule

Action:

6268 Resumes running the schedule. Any traffic on the console will cause the schedule to stop. This command allows resumption of processing.

Arguments:

Schedule Name: S0= ... S9=

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

9.2.5 STAR

Scheduling Control.

Table 22: Start Scheduling

Command sample	**Description**
STAR *start time*	run selected schedule after specified time

Action:

6307 Enables the *mod* schedules to start running after the specified time.

Arguments:

Starting Time: *HH*:*MM*:*SS*.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

This command allows the unit to remain quiet, emitting no RF, until the hunt actually begins.

The schedule scan is suppressed while the specified time is less that the system time. The test is performed against the internal system time, in terms of ZULU time. The time is truncated to a single day (i.e. 86400 seconds) for the comparison, so there is no point in giving time beyond the normal 24 hour day.

If the **EPOC** command, if it is used, should occur prior to this command so that the local time zone is taken into account.

9.2.6 IDLE

Table 23: Idle

Command sample	**Description**
IDLE	Stops all schedules

Action:

Clears the run flag for all schedules.

Arguments:

None.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

9.2.7 STAT

Table 24: System Status

Command sample	**Description**
STAT	System Status with module compile times
STAT I	System Status with module compile times

Action:

Displays current system status.

Arguments:

"I" to list the compile dates for all the modules that make up the Fox Transmitter software suite.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

9.2.8 CONF

Table 25: Hardware Configuration

Command sample	**Description**
CONF	Configure Hardware Subsystems

Use the **CONF** command with no arguments to dump the keywords available to this command. The **STAT** command also shows the configuration flags.

Action:

Sets the hardware configuration bits.

Arguments:

Keyword(s).

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Table 26: Hardware Configuration Flags

Command flags	**Description**
ICS525	Select ICS525 synthesizer
ICS307	Select ICS307 synthesizer
SI5351	Select SI5351 clock synthesizer
SA818	Select SA818 transmitter
DRA818	Select DRA818 transmitter
EXTERN	Select external transceiver
CTL	DRA818/SA818 Power Down Enable
PTT	Carrier Enable (external transceiver)
TX_ENA	Transmit enable
DB_PWR	Daughterboard Power Switch
TONE	Square Wave Tone Gen enable
PWMH0	Audio Generator Enable
AUDIO	Audio waveform device select
ICS_PD	ICS chip power down (ICS525/ICS307)
VCMO	VCMO Enable (102-73161)
5MON	ANALOG: Enable 5 volt monitor
SWIT	ANALOG: Enable ext switch monitor
PHOTO	ANALOG: Enable ext photo monitor
IMON	ANALOG: Enable batt current monitor
VMON	ANALOG: Enable batt voltage monitor
I2COFF	Suppress I2C traffic
CWTIM	Emit CW timing report
SCHED	Scheduler diagnostics
DEBUG	Enable diagnostics
2MA	SI5351 clk drive 2mA
4MA	SI5351 clk drive 4mA
6MA	SI5351 clk drive 6mA
8MA	SI5351 clk drive 8mA
CLK0	SI5351 enable CLK0, direct
CLK1	SI5351 enable CLK1, buffered
CLK2	SI5351 enable CLK2, LVDS
CLEAR	Clear configuration flags
BMON	Select the battery voltage monitor voltage divider
VOICE	Select the memory device for audio storage
T0= ... T5=	Set timing slots for message transmission

The 102-73161-7 hardware and the 102-73161-12 hardware are deprecated. This software will not operate correctly with these hardware revisions.

The 102-73161-25 hardware maintains the control lines at the same polarity as the 102-73181 revisions, so the *ICS525* flag allows the ICS525 to be loaded.

The 102-73181-0 hardware makes use of the *ICS307* flag to configure for this synthesizer. None of these boards were built and the software has not been tested to work with this hardware revision.

The 102-73181-5 and 102-73181-10 hardware operate identically as far as the zNEO is concerned. Either of these hardware revisions are selected using the *SI5351*, *SA818*, of *DRA818* flags.

ICS525 SubCommands

RF Synthesizer select.

For the 102-73161-25 boards, set the configuration flags to operate with the ICS525 device.

ICS307 SubCommands

RF Synthesizer select.

Not implemented (102-73181-0).

SI5351 SubCommands

RF Synthesizer select.

For the 102-73181-5 and later boards, set the configuration flags to operate with the SI5351 device.

Additional selection for the clock output drive and clock select are necessary.

SA818/DRA818 SubCommands

RF Synthesizer select.

For the 102-73181-5 and later boards, set the configuration flags to operate with the SA818 and DRA818 transceiver module.

Currently the SA818 and DRA818 are treated identically.

EXTERN SubCommands

RF Synthesizer select.

For all boards, set the configuration flags to operate with an external transceiver.

Although earlier revisions had the external transceiver connection, these early revisions do not implement serial control correctly so that capability. Serial control of the handie-talkie requires 102-73181-10 or later circuit board.

nMA SubCommands

SI5351 clock output drive.

This is a selection command, only one of the SI5351 clock drive values is active.

CLKn SubCommands

SI5351 clock output select.

This is a selection command, the software only allows one clock output to be active.

Although the hardware can drive all output at once, the software only enables one of the three clocks. Take note that the output clock selection must match the hardware configuration.

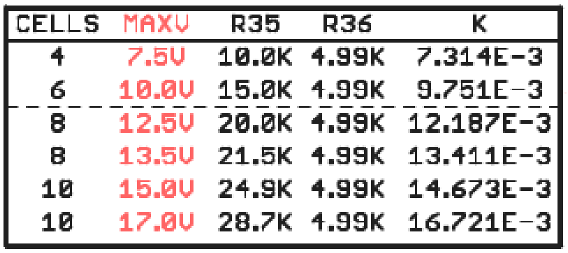
For example:
When using the LVDS amplifiers, 102-73161-29 or 102-73181-35, the CLK2 output must be selected or the LVDS driver on the main board will not receive a clock.

CLEAR SubCommand

This sub-command clears all the configuration bits to zero. This does not affect the SI5351 configuration fields that control the SI5351 clock drive nor does it affect the SI5351 output clock channel select.

BMON SubCommand

This sub-command informs the software of the resistor values used in the battery voltage divider.



CELLS	MAXV	R35	R36	K
4	7.5V	10.0K	4.99K	7.314E-3
6	10.0V	15.0K	4.99K	9.751E-3
8	12.5V	20.0K	4.99K	12.187E-3
8	13.5V	21.5K	4.99K	13.411E-3
10	15.0V	24.9K	4.99K	14.673E-3
10	17.0V	28.7K	4.99K	16.721E-3

Figure 23: BMON values (keyword in RED)

The keywords in the **MAXV** column are used to indicate to the software the specific resistor divider value (for R35) present on the circuit board.

The default configuration uses a 15.0K resistor for R35 which is intended for a 6-cell AAA battery pack. Using the AAA cell holder and housing indicated in the drawings, the battery pack fits.

If a larger pack is used, R35 can be swapped out for the indicated values to accommodate a higher capacity pack or a higher input voltage.

VOICE SubCommand

Selects either the FRAM device (U3) or the FLASH device (U12) for storing audio data.

DB_PWR SubCommand

Currently, this command affects how the **CHRP** command controls the DB_PWR control net. When the flag is set, the CHRP command will switch the DB_PWR net off when there is no audio tone, effectively interrupting the carrier between chirps.

This flag need not be set with the 102-73181-28 RF amplifier as the TX_ENA net switchs carrier on and off on the 102-73181-28 board.

T0=nn Timing SubCommand; RF subsystem power enable

Sets (or resets) the time for the message transmission T0 slot.

This delay is in the *radio_control.c* module.

The **DB_PWR** signal is asserted to turn on U81 to apply power to the RF daughter board.

The DRA818/SA818 require some time to collect its thoughts before any command traffic will be processed.

T1=nn Timing SubCommand; RF subsystem powerup

Sets (or resets) the time for the message transmission T1 slot.

This delay is in the *cmd_message.c* module.

The **CTL** signal is asserted in this state (this appears as **PD'** on the SA818/DRSA818 daughter board).

It occurs right after the T0 delay has occurred.

The DRA818/SA818 require some time to recover from a power-down state before any command traffic will be processed. The standard timing value for this RF module is 1500 mSec and it may need to be extended for some instances of the module.

Do keep in mind that this delay directly moves when the RF section starts producing carrier. If there is a mix of RF modules, this delay must be accounted for; with the easiest approach being to use the same T1 delay throughout the group.

Using matching timing parameters throughout the group will, in effect, shift all messages from the group of fox transmitters by the same time. This will be most evident when a mix of SA818/DRA818 RF modules, and SI5351 RF clock synthesizer are used. The SI5351 will not require the long delay needed by the SA818/DRA818 RF modules, but the long delay will not interfere with SI5351 operation.

T2=nn Timing SubCommand; RF enable

Sets (or resets) the time for the message transmission T2 slot.

This delay is in the *cmd_message.c* module.

The **TX_ENA** signal is asserted in this state (this appears as **PTT*** on the SA818/DRSA818 daughter board).

This delay occurs immediately after RF appears on the antenna connector.

T3 Message Delivery

There is no timing delay associated with this timing state.

This is the state where our message is sent.

Time spent in this state is determined by the message content and the code generator settings (i.e. see CPWM command).

T4=nn Timing SubCommand; RF disable

Sets (or resets) the time for the message transmission T4 slot.

This delay is in the *cmd_message.c* module.

The **TX_ENA** signal is deasseerted in this state.

This is a (typically) short delay after the last of the outgoing message has been sent. Although the output channel is not pipelined, and we could simply disable RF after the last message event, this delay is provided to enforce some quiet time after the last message audio is sent.

This simply provides us a means of avoiding a garbled ending to our message.

T5=nn Timing SubCommand; RF teardown

Sets (or resets) the time for the message transmission T5 slot.

This delay is in the *cmd_message.c* module.

The **CTL** and **DB_PWR** signals are deasserted in this state.

This delay occurs prior to steps taken to bring the RF subsystem into a low-power state.

9.2.9 TOYC

Table 27: TOY Clock Charge

Command sample	**Description**
TOYC	

Action:

Sets up the DS1672 charge control in the same manner as the TOYE command.

Control register patters are inspected and the write is suppressed if the control bits are not as expected. This checking is to avoid writing an incorrect time into the 32 bit time counter,

Arguments:

Charge rate:

250 enable the 250Ω charging resistor
2K enable the 2KΩ charging resistor
4K enable the 4KΩ charging resistor
NONE disable charging circuit

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

The 102-73181-5 boards added a circuit to supply current to the DS1672 from the main battery to maintain the clock battery. For these revisions the appropriate argument to this command is "**NONE**"

9.2.10 TIME

Table 28: Time management command

Command sample	**Description**
TIME *unix-time*	load time field into DS1672
TIME	load system time from DS1672

Action:

Set the system time and TOY clock time.

Arguments:

Time String.

6901 **Returns**:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

With no argument, the DS1672 is read and its contents are written to the system time field. There is no provision to attempt to deal with sub-second level time resolution, the transmitter system ends up withing a second or two of the actual time.

If a time string is present, it is copied to the system time field and the DS1672 time field is updated.

TOY is an acronym for *TIME of YEAR*.
The DS1672 is a 32 bit counter that is incremented every second.
The system time is kept as a 32 bit integer, again being incremented once per second.

9.2.11 D525

Table 29: ICS525 management command

Command sample	**Description**
D525	Dumps the ICS525 Divisors

Action:

ICS525 setup diagnostic.

Arguments:

6942 None.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Right now, no arguments are expected.

This command will expand to deal with ICS525 management.

9.2.12 EPOC

Table 30: Time management command 3

Command sample	**Description**
EPOC *argument*	

Action:

Set the timezone offset.

Arguments:

Offset in hours.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

The TOY clock is expected to be loaded with **UT** or **truncated UT**.
The time display routine shows only hours minutes and seconds.
This offset is added to the current system time in the time display routine to shift the displayed time to local time.

Here in the central time zone, the argument is **-5.0** in the summer and **-6.0** in the winter.

In the example sequence, we set the timezone offset, which you may think of as the local day epoch, prior to loading time from the TOY clock. The time displayed on the debug console will then be expressed in local time.

Note: **truncated UT** would be:

```
        system_time  = Unix_Time % 86400;
        system_time += 86400;
```

9.2.13 CALL

Table 31: Setup Callsign

Command sample	Description
CALL	FCC assigned callsign

Action:

The supplied text is saved in the callsign variable.

This callsign is sent at the end of every message.

7047 **Arguments**:

Callsign.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

This provides a single instance of the callsign.

This callsign may be substituted, as in **CODE** <CALL> or **TALK** <CALL>

9.2.14 NAME & NICK

Table 32: Setup Nickname

Command sample	Description
NAME	tactical callsign
NICK	tactical callsign

Action:

The supplied text is saved in the name/nickname variable.

7091 **Arguments**:

Nickname.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

This provides a local alias for the transmitter.

This nickname may be substituted, as in **CODE** <NAME> or **TALK** <NAME>

9.2.15 TONE

Table 33: CW Tone Control

Command sample	**Description**
TONE *Hz*	

Action:

7123 Sets the audio tone frequency.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

The specified frequency, in KHZ is approximate.

9.2.16 CWPM

Table 34: CW chipping rate control

Command sample	**Description**
CWPM *wpm*	Set the basic CW delivery rate
CWPM *wpm bit char word sentence*	Set all of the code generation parameters

Action:

This command sets the code generation timing parameters.

Arguments:

Chipping parameters from the table:

Table 35: Chipping Parameters

Num	**Argument**	**units**	**Description**
1	WPM	*words/minute*	Basic code delivery rate
2	inter-bit	*chips*	Nominal Value 1
4	inter-character	*chips*	Nominal Value 3
3	inter-word	*chips*	Nominal Value 5
5	inter-sentence	*chips*	Nominal Value 7

There are two special cases that may be specified with this command.
The chipping rate may be increased by 1 WPM using "++" as the only argument.
Conversely using "–" will decrease the chipping rate by 1 WPM.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

The *chip* referred to above is the time allocated to a **dit** in the code message. A **dah** is allocated 3 **dit** times.

In addition to the basic chipping rate (expressed in terms of WPM), the spacing between tones may be adjusted as needed using the 2nd. through 5th. fields.
Use a value of *0* to leave parameters unchanged and a value of *-1* to set the field to the nominal or default value.

The **STAT** command will display the current settings.

9.2.17 FREQ

Table 36: Transmit carrier frequency control

Command sample	**Description**
FREQ *freq*	Operating Frequency

Action:

Save the Transmitter Frequency.

Arguments:

Frequency, in MHz.

The action performed depends on the transmit element selected in the **CONF** command.

SA818/DRA818 Transceiver Module

The frequency string is saved for later use.

When the **BEGN** command configures the transceiver, the frequency string is sent to the RF module to set the frequency.

The only sanity check is to check that the requested frequency is within bandplan allocations.

SI5351 Clock Synthesizer

The frequency string is saved for later use and used to search the internal frequency table for a matching frequency.

Assuming you have entered a valid frequency, the SI5351 setup patterns are extracted and saved for use when the **BEGN** command configures the SI5351.

The result of the lookup are visible in the **STAT** command reports and a failed lookup is reported in the *stsxx,xx** report.

9.2.18 5351

Table 37: SI5351 Control

Command sample	Description
5351	register key and parameters

Action:

SI5351 setup.

Arguments:

Keyword and hexadecimal parameters

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Table 38: SI5351 Register Parameters

key	Arguments	Description
HELP		This help file
DUMP		Dump SI5351 Registers
SDUMP		Dump ALL SI5351 Registers
TABLE		Dump SI5351 Setup Tables
RESET		Reset SI5351
LOAD		Load SI5351 Registers
TEST	<P1>,<P2>,<P3>	Test 5351 divisor values
CAP		Set Xtal load capacitors (6PF, 8PF, 10PF)
XTALT		Connect XTAL oscillator to CLK0 pin
I2CT		Test I2C transactions to SI5351
I2CR		Test I2C Read transactions to SI5351
I2CW		Test I2C Write transactions to SI5351
FREQ	<f>,<d>,<o>	Save Frequency, Divisor and Offset
PLLS	<P1>,<P2>	Save MSNx register values (P3=PLL3_DEFAULT)
PLLS	<P1>,<P2>,<P3>	Save MSNx register values
MS	<P1>,<P2>,<P3>	Save MS register values

HELP Help Text

Dumps the help text through the serial channel.

DUMP Register Dump

7339 Dump the SI5351 registers through the serial channel.

SDUMP Full Register Dump

7348 Dumps the entire 256 byte address space of the SI5351 through the serial channel.

TABLE Frequency Table Dump

7357 Dumps the frequency table through the serial channel.

RESET SI5351 PLL Reset

7368 This is supposed to assert the reset bits in the SI5351 that reset the two internal SI5351 PLLs. V3.56 doesn't get it right!

LOAD Loads the SI5351 register.

Loads the SI5351 with the stored patterns.

7380 Used to test a configuration that has been set using the FREQ, PLLS, and MS sub-commands.

TEST Test SI5351 MSNA divisor values.

Picks up the MSNA/MSNB divisor values from the command line, loads the SI5351 registers (all of them), and starts sending a CW "HI" message until a keyboard character arrives.

7396 Used to quickly test the MSNx divisor values.

The transmitter (i.e. the RF generator) is shut down when a keystroke comes in.

CAP Load Capacitor

Set the SI5351 crystal oscillator load capacitors.

7409 Loaded into the SI5351 when the LOAD sub-command occurs or when the **BEGN** occurs.

XTALT Diagnostic

7419 For diagnostics, the crystal oscillator output is connected to the CLK0 output pin.

I2CT I2C transaction test

7428 Generates continuous I2C transactions addressed to the SI5351.

I2CR I2C transaction test

7437 Generates continuous I2C read transactions addressed to the SI5351.

I2CW I2C transaction test

7446 Generates continuous I2C write transactions addressed to the SI5351.

FREQ Frequency Table Load

Saves the parameters to RAM for later use.

These strings are, in effect, simply documentation fit for human consumption. They contents are not used to load the SI5351.

PLLS Multisynth 1 parameters

These are the P1, P2, and P3 register values.

MS Multisynth 2 parameters

These are the P1, P2, and P3 register values.

9.2.19 BEGN

Table 39: Begin Message Traffic

Command sample	**Description**
BEGN	
BEGN *SILENT*	

Action:

Begins message transmission.

Arguments:

SILENT modifier.

This modifier suppresses the code ID that is sent as part of the BEGN sequence. This in a non-FOX related modifier, not to be used when transmitting over the air.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Enables the RF section.
When configured for an external device, the on-board RF sections would be left disabled.

Once the RF subsystem is active, the system sends a **CQ** call and an initial callsign.

The current system clock is saved for use by the *DONE* command.

9.2.20 CODE

Table 40: Generate Morse Code

Command sample	**Description**
CODE *text*	

Action:
Sends a CW message.

Arguments:

Message.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

The message text is sent at the specified word rate (see the **CWPM** command).

Multiple messages may be sent, one after other, to overcome the limited size of each record in the FRAM.

The chipping rate may also be altered at any time. That is to say that **CODE** commands and **CWPM** commands may be freely intermingled.

9.2.21 TALK

Table 41: Generate Audio

Command sample	**Description**
TALK *clip-name*	

Action:
Send a voice message
Arguments:

File Name.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Voice waveform data, stored in the FLASH, is used to load a PWM channel in the zNEO to generate a PWM signal to the RF subsystem.

The sample rate is set by a control field in the TALK directory. The the word size is fixed at 8 bit unsigned.

Aliasing: Aliasing is allowed in the **TALK** command to allow callsign and nickname substitution. The audio file name must match the stored callsign or nickname as the lookup is a simple text substitution.

Waveform data is stored in the FLASH rather than in the FRAM assuming that the waveform data will not change frequently. The FLASH device is also much larger than the FRAM device and considerably cheaper.

The **TALK=** directory entry has the starting location in the FLASH memory along with the size of the audio file and its sample rate. Sample rates of 4K/sec, 5K/sec, and 8K/sec are implemented.

9.2.22 WAIT

Table 42: Simple Wait

Command sample	**Description**
WAIT	Wait specified time (in seconds)
WAIT SWITCH x	Wait for switch input state
WAIT PHOTO	Wait for Photocell input change

Action:

Waits for specified time.
Waits for switch state.
Waits for photocell change.

Arguments:

Time in seconds (decimal).

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

This is a simple delay. Time specified in decimal seconds. Time specification must be between 1/10th. and 60 seconds.

V3.53 adds the *SWITCH* modifier to check the status of front panel switch that is connected through the J6 connector.

The *PHOTO* modifier is also part of this addition. This modifier keeps a running average of what the photocell sees and releases when it changes.

9.2.23 CHRP

Table 43: Chirp Emulator

Command sample	Description
CHRP	

Action:

Chirping tracker emulation.

Arguments:

Audio tone frequency (KHz).
Chirp period (seconds).
Chirp duration (seconds).
Repeat count.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

This command emulated a wildlife tracker that turns on RF for a short period every few seconds. The carrier is removed between chirps.

The meaning of *period* matches its meaning in the scheduling command. That is to say the chirp will occur every *period* seconds. The chirp tone duration is set by the *Chirp duration* parameter.

Argument 1: frequency

Audio tone frequency in Kilo-Hertz..

A value of 0.0 disables the modulation signal resulting in an unmodulated carrier being sent.

The audio tone frequency here does not affect that set by the **TONE** command.

Argument 2: seconds

Carrier Duration.

Specifies the time (in seconds) the chirp is sent. Carrier is active during this period.

Argument 3: seconds

Inter-carrier duration.

Specifies the time (in seconds) between chirps. Carrier is suppressed during this period.

Argument 4: count

7794 Repeat count.

The chirp is repeated this many times.

9.2.24 DONE

Table 44: Done with message traffic

Command sample	Description
DONE	
DONE *SILENT*	

Action:

End of message traffic.

Arguments:

SILENT modifier.

7844 This modifier suppresses the code ID that is sent as part of the BEGN sequence. This in a non-FOX related modifier, not to be used when transmitting over the air.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Done with message traffic.
The ending callsign message is sent and the RF subsystem is disabled.

The ending message consists of the station callsign (see the **CALL** command) to satisfy FCC rules and a CW *SK SK SK*.

9.2.25 BATC

Table 45: Battery Report CODE

Command sample	Description
BATC	<modifier>,<key>,<V-limit>

Action:

Battery Report Request.

Arguments:

Modifier.
Channel Key.
Battery Voltage Limit.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Table 46: BATC Modifiers

Modifier	Description
A	A/D sample collected after BEGN
B	A/D sample collected before BEGN
E	Encode value using T/E

Table 47: BATC Keywords

Channel Key	Description
V	Battery Voltage
I	Battery Current
R	Regulated 5V rail

This command generates a CW battery report. The command should occur between **BEGN** and **DONE** commands (or you won't hear it). Although the battery status is available through the **STAT** command, battery status isn't visible when transmitting.

The *modifier* must be the first character after the BATC command and it is optional. The **A** and **B** modifiers control when the A/D is sampled. If the modifier is omitted, the sample data is taken when there is RF carrier present.

The **E** modifier changes the encoding of the engineering value reported over the air. When the **E** modifier is omitted, the voltage or current will be reported using numerics (so you have to be able to hear CW to understand it). The report can also be encoded where the first digit is a series of "T"s (i.e **dah**s) and the second digit is a series of "E"s (i.e **dit**s).

As with all CW generating commands, the **CWPM** command can be used around this **BATC** command to adjust the speed to your liking.

The keywords, which occur after any modifier character, select the channel to report on. Two voltages and one current may be reported.

Of primary interest is the battery condition, that is the voltage and current draw. The voltage being of interest to know if the station can operate for the duration of a hunt (or needs a battery change).

The current simply to see that the station is operating normally or if a fault condition needs to be investigated. Since the primary regulator, for the 5 volt rail, is a switchmode device, the current is inversely proportional to the battery voltage.

Finally, the regulated 5V rail is just there, if you want to look at it. If it's low, there is a good chance the station is about to go quiet because the battery can't support the load. If the 5V rail is high, the 3.3V regulator is probably about to overheat.

The *battery voltage limit* is used to alter the battery voltage message. This field is ignored for the other two channels.

This set the point where the battery voltage message change from *BATC HI HI nn.n* to *BATC SOS SOS nn.n* to indicate that the battery voltage is getting too low and the station may soon go off the air.

Note on command timing:// The use of the **E** modifier makes the execution time of this command a bit unpredictable. The number of **T** and**E** CW elements are voltage dependent.

9.2.26 BATV

Table 48: Battery Report VOICE

Command sample	Description
BATC	

Action:

Battery Report Request.

Arguments:

8026 Keywords.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

This has all the same modifiers and keywords used by the **BATC** command with the exception of the **E** modifier. The **E** modifier is of no use when generating voice traffic so it is ignored if present.

The *battery voltage limit* is also ignored. This function may be added in future releases.

9.2.27 MODS

Table 49: Scheduling control, MOD

Command sample	**Description**
MODS *n period offset*	load schedule

Action:

Sets a schedule.

Arguments:

Schedule.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Argument 1: n

Schedule number.

Runs from 0 through 9, allowing ten schedules to be present in the schedule.

Argument 2: period

Scheduling period, expressed in seconds or minutes and seconds.

Argument 3: offset

Scheduling offset, expressed in seconds or minutes and seconds.

The scheduling period may be expressed as seconds, should that be most convenient. To specify period or offset in seconds, simply place the number of seconds in the command. Fractional numbers should not be used as the scheduler granularity is limited to seconds.

If you find using minutes and seconds more convenient, encode them with a colon as a separator. The presence of the colon triggers the correct handling of the period to offset specification.

The scheduling period is synchronous with system time. That is to say the scheduling period offset is the remainder when the system time is divided by the scheduling period.

A scheduling period of 5 minutes starts on the hour, and every five minutes thereafter. Matching scheduling periods (in multiple stations) are all aligned.

The scheduling offset is the offset into the scheduling period where this particular schedule begins operation. Each station in a group has the same scheduling period. Each station in a group has a unique scheduling offset. The scheduling offsets differ by the time allocated to each station.

As an example a normal foxhunt with a 5 minute cycle would be scheduled like this:

Table 50: Typical Schedule

Unit	**Period**	**Offset**	**Notes**
1	300	0	message time limited to less than 60 seconds.
2	300	60	60 seconds limit
3	300	120	60 seconds limit
4	300	180	60 seconds limit
5	300	240	60 seconds limit

9.2.28 MODC

Table 51: Schedule clear

Command sample	**Description**
MODC *Sn*=	

Action:

Clears a schedule.

Arguments:

Schedule Name (with the trailing equal sign)

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Argument 1: n

Event number.

The event number is expressed the same manner as with the MODS command above. This command allows all schedules to be stored and edited in one file. Individual setups, where the callsign is set, can then clear all the *other* schedules.

9.2.29 TALK Filesystem directory commands

These pseudo-commands reside in the *Configuration Command File System* but are not processed by the normal command processor. These command strings define the location and length of the audio clips in the TALK Filesystem. The **TALK** command uses these pseudo-commands to locate the audio waveform data in the FRAM.

When the audio clip is a properly formatted RIFF/WAVE file with a header, only the start address in FLASH is needed. Additional information is extracted from the header.

Name Audio File Name This is a simple file name. Text processing when loading the directory forces the filename to be uppercase.

> An underscore is allowed in the filename.
>
> Although parameter substitution is performed for arguments to a TALK command, this is not allowed in these directory entries.
>
> Filename length is no specifically limited, but all the information in the directory entry must fit into the 31 byte records size limit.

Start Audio File start address This is the decimal starting address of the audio file.

> A start address is always required. The RIFF/WAVE file header does not have any content that could be used to perform a search.

Length Audio File length This is the decimal length of the audio file. It is the number of 8 bit data samples that will be processed by the **TALK** command..

> A properly formatted RIFF/WAVE file header eliminates the need to supply this parameter.

Rate Audio File Sample Rate This field is a key to tell the **TALK** command the audio sample rate of the file. The **TALK** command uses this field to set the SPI clock rate used to read from the FLASH device. The SPI clock rate then dictates the update rate of the PWM control register.

> A properly formatted RIFF/WAVE file header eliminates the need to supply this parameter. This parameter is restricted to these same three rates.
>
> There are currently 3 valid keys for this field. A key of "**4K**" set a sample rate of 4KHz. A key of "**5K**" set a sample rate of 5KHz. A key of "**8K**" set a sample rate of 8KHz.
>
> For over-the-air voice, a sample rate of **4K** or **5K** may be used. A sample rate of **8K** will cause the RF section to produce too much deviation and produce excessive bandwidth.

9.2.30 ESAV text

Table 52: FRAM control ESAV

Command sample	Description
ESAV	

Action:
Save a command into FRAM-volatile memory.

Arguments:
Command text.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

9.2.31 EDMP text

Table 53: FRAM control EDMP

Command sample	Description
EDMP	

Action:
Dump FRAM-volatile memory records.

Arguments:
Search Key.

Returns:

sts01,nn* records with active FRAM records.

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Scanning of the FRAM records stops when an empty record is encountered.

The **EZER** command makes subsequent records *invisible*!

9.2.32 EDID

Table 54: FRAM control EDID

Command sample	**Description**
EDID	

Action:

Dump FRAM/FLASH JEDEC ID.

Arguments:

None.

Returns:

sts01,nn* records for the FRAM device.

sts01,nn* records for the FLASH device.

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

9.2.33 ERAS

Table 55: FRAM control ERAS

Command sample	**Description**
ERAS *n*	erase record (i.e. REM-)
ERAS *DEV*	erase device
ERAS *CMD*	erase device

Action:

Zero FRAM record.

Arguments:

Record Number or keyword

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Numeric Record Number

Specifying the record number rewrites the specified record with a **REM-** command.

This leaves the remainder of the FRAM file system visible.

DEV Entire Device

Erase the entire FRAM device.

9.2.34 EZER

Table 56: FRAM control EZER

Command sample	**Description**
EZER *n*	erase record (ZERO)

Action:

Erase a single record from FRAM.

Arguments:

Specifying the record number zeroes the specified record.

This leaves the remainder of the FRAM file system *visible*.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Use the ERAS command to change the record to **REM-**, leaving following records visible.

This command zeros the FRAM record out, making any records that follow *invisible* until an **ESAV** command rewrite the record.

9.2.35 ETAB

Table 57: FRAM/FLASH table dump

Command sample	Description
ETAB	FRAM & FLASH device table dump

Action:

Dump FRAM/FLASH JEDEC-ID table.

Arguments:

none.

Returns:

sts01,nn* records with device information.

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

The entire device ID table is dumped.

Each detail line has the following:

Table 58: FRAM/FLASH device Table

Column	Contents
Write-Mode	write/erase strategy
JEDEC-ID	device ID bits
Size	device size, in bits
Page	write page size in bytes
Sctr	sector erase size in bytes
Manufact	Device manufacturer
Device	Device part number

Column: Write-Mode

A field indicating to the software the write and erase strategy for this device.

The typical FLASH device will be of the **FLASH_PAGE** type, indicating that the device can deal with a multi-byte write.

The typical FRAM device will be of the **FLASH_FRAM** type, indicating the device can deal with any length write and does not require device erase to update records.z

A not-so-useful device, FLASH_AAI, can write 2 bytes at once. Probably too slow to be at all useful for this project.

Column: JEDEC-ID

This is the 3-byte sequence used to recognize the device.

Some devices may appear twice, with slightly different patterns to accommodate the chip reading method (1st. byte is 0x7F).

Column: Size

This is the device size, in bits.

FRAM type devices tend to be more expensive as size increases. We use a reasonable size device, typically 64Kb to 256Kb, to store commands.

Large capacity FLASH type devices tend to be less expensive. These type are used to store waveform data.

Column: Page

This is the write page size in bytes.

FRAM device will have a page size of 1. This class of device writes a byte-at-a-time at wire speed. This makes for easy erasing of a single command record (32 bytes).

FLASH device need to have a page size of at least 32 bytes. This lower limit is imposed by the expecte4d Intel-HEX record size that can be handled by the command parser.

Column: Sctr

This is the sector erase size in bytes. A size of zero indicates the device functions and a non-volatile RAM device.

Currently, the FLASH device (U12) is managed as a single *lump* of memory. It is erased as a unit.

Sector erase may be fully implements at some point to improve the flexibility of the system, but not now...

Column: Manufact

Device manufactured by.

This is simply obtained from the device data sheet and entered in the table.

Column: Device

Device part number. This should match package markings.

This is also obtained from the device data sheet and entered in the table.

9.2.36 HERA

Table 59: FLASH control HERA

Command sample	Description
HERA *size*	erase record
HERA *size start*	erase record
HERA DEV	device erase

Action:

Erase region of FLASH device.

Arguments:

Region Specifier.

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

This command is **in**sensitive to the **CONF VOICE** setting (see section 9.2.8 on page 105). You can use the **HERA** command to bulk erase audio data in the FRAM device.

Also keep in mind that chip erase time for many flash devices are quite long, exceeding 100 seconds for some. This command does not wait for the erase operation to finish, it simply returns after sending the **chip erase** command to the device.

This makes the flash device look *dead* until the **chip erase** operation has finished. Sending any flash commands will return a **BUSY** message until the device reports it is ready.

9.2.37 HDMP

Table 60: FLASH control HDMP

Command sample	Description
HDMP *length start*	dump 32 byte records

Action:
Dump an area of the FLASH device.

Arguments:
Length (in 32 byte lines) and start address.

Returns:

:20 .... (Intel HEX dump records)

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

The hex commands (HDMP HERA and :xxxx) are sensitive to the VOICE configuration setting (see the CONF command in section 9.2.8 on page 105). This feature should be unused with the 102-73181 boards with dual serial memory. This feature allows the 102-73161-25 board to store audio clips in the single serial memory device.

9.2.38 H56K

Action:
Switch bit rate to 57,600 bits/second.

Arguments:
None.

Returns:

RDY00,00* response with current stack pointer value and current system time. Operating at the new bit rate.

The commands to speed up the audio loading process.

First, switch the target over to 57,600 b/s.

```
H56K
sLask.jcn
```

The STS and RDY response are, of cource, garbled due to the mis-matched bit rates.

Next switch the monitoring terminal over to match the new bit rate of 57,600 b/s.

```
.../halo_term -b57600 -STACH
```

8848

And then we can proceed to download the audio HEX file at the new, higher, rate. We trim the delay between each line sent to the target to a bit less than 100 milliseconds (i.e. the **-c90** on the call line).

```
.../fox_simple -b57600 -STACH -c90 -t10 -ffox_73181_rxxa.hex
```

The higher bit rate reduces the transmission time, of cource, and reduces the time the target spends sending the shortened RDY message.

Do keep in mind that the target serial channel is buffered on the input side (the entire input line is buffered by the ISR) 8856

9.2.39 :hex

Table 61: Intel HEX Record Load

Command sample	**Description**
:hex *record*	Intel HEX record up to 32 bytes long

Action:

Load FLASH device.

8899 **Arguments**:

Intel HEX record (up to 32 bytes of data)

Returns:

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

The input buffer is examined for the leading colon character before normal command processing. An Intel HEX Record is diverted and processed as FLASH data.

Sample audio hex record:

```
:02 0000 04 0000 FA
:20 0000 00 5249464650100000574156456666D74201000000001000100A00F0000A00F0000 4F
:20 0020 00 01000800646174612B10000080808180807F8080808080808080808080808080 E2
```

This example text comes from the audio utility. The embedded spaces are added by the audio utility to make looking at the record a bit easier on the eyes.

This command is sensitive to the **CONF VOICE** setting (see section 9.2.8 on page 105).. See comments in section 9.2.37 on page 138.

> When configured to use the FRAM for voice storage, this command will happily write to the command area (i.e. the bottom of FRAM where commands/sequences are stored). Memory allocation is strictly manual.

File load accommodations:

> The status report generated when encountering an *Intel HEX Record* is abbreviated in an attempt to minimize traffic on the command channel. All that is reported is a **RDY00,00*** to indicate we are ready for the next record.
>
> All **sts00,00*** and **STS00,00*** reports are suppressed.

9.2.40 HALT

Table 62: HALT Instruction

Command sample	Description
HALT	

Action:

Execute the zNEO **HALT** instruction until keypress.

Arguments:

None.

Returns:

Stream of periods to indicate the zNEO is fielding interrupts.

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

The puts the zNEO HALT instruction in a loop to verify that the timer used to generate the 100Hz scheduling interrupt s functioning.

Any keystroke returns control to the main loop.

9.2.41 STOP

Table 63: STOP Instruction

Command sample	**Description**
STOP	

Action:
Execute the zNEO **STOP** instruction. This will hang the processor!

Arguments:
None.

Returns:

Nothing as the zNEO stops executing instructions.
A hardware reset is required to recover.

zNEO stops (hangs the system).

The command executes the zNEO STOP instruction.

As there are no hardware provision to bring the processor out of a STOPped state, this will hang the system.

This testing command may be removed from a future software update.

9.2.42 TEST

Table 64: Test Suite

Command sample	**Description**
TEST	Misc. test commands

Action:
Pass control to test handler

Arguments:
Text.

Returns: (typically)

sts01,nn* response with status and command execution time.

RDY00,00* response with current stack pointer value and current system time.

Table 65: Test Suite Tests

Command flags	**Description**
none	Help List
HLP	Help List
DIT	CW Interrupt Test
CWR	Chipping Rate
SPI	SPI path test
I2C	I2C path test
LED	LED/PTT
U13	U13 mux
TXE	TX_ENA (4 sec cycle)
BAT	Battery Monitor
TXD	Data channel to daughter board
GPX	Test GPIO Bits
CFG	Dump fox_config struct
NEO	zNEO port bit dump

HLP Help List

DIT CW Interrupt Timing

CWR CW Chipping Rate Test

SPI Activity on SPI pins

I2C Activity on I2C pins

LED Toggle Transmit Enable signal

U13 Serial Channel 2 Rx MUX Test

TXE Toggle Transmit Enable signal

BAT Continuous Battery Monitor

TXD Serial path to daughter board

GPX Power Control Bits toggle.

CFG Dumps configuration information.

NEO Dumps all the GPIO control register bit patterns

9.3 Sample Sequences

Example code for setting up a FOX Transmitter.

9.3.1 Initialization

This code runs following reset, hence the *INI=* tag.

Table 66: Sample Sequence 1

esav INI=TIME
esav INI=CALL KC0JFQ
esav INI=EPOC,-5.0
esav INI=NAME FOX20
esav INI=CONF DRA818
esav INI=MODS S1 300 0
esav INI=MODS S2 300 150
esav INI=MODS MAS 5 4
esav INI=TONE 1.6
esav INI=CWPM 25
esav INI=FREQ 144.150
esav INI=STAT

9.3.2 TIME

Time Set. This loads the 32 bit time from the TOY clock into the system time field. This is required in order for multiple units to operate synchronously.

9.3.3 EPOC

Time Zone Set. Somewhat akin to establishing an epoch from which time starts.

This is **not** required in order for multiple units to operate synchronously, it simply provides for the command echo to reflect local time.

9.3.4 CALL/NAME

Sets the callsign and nickname for the CQ and SK messages the precede and follow message traffic.

9.3.5 CONF

Tells software the RF subsystem in use. traffic.

9.3.6 MODS

These are the modular schedules. Up to 10 schedules may be processed with this revision of the software. Although it would be possible to operate more schedules, in practice processing additional schedules requires more battery power to support the additional CPU cycles.

9.3.7 TONE

Sets the audio tone frequency.

9.3.8 CWPM

Sets the code chipping rate to 25 WPM.

9.3.9 FREQ

Selects the transmitter frequency. Although any frequency may be specified in this command, only a limited set of frequencies can actually be generated. The software selects the closest available frequency.

9.3.10 STAT

Displays the system status on the USB serial port.

9.4 Announcement

This code also runs following reset, the *ANN=* tag keeps the RF activity separate from the initialization.

This group of commands are run following the *INI=* commands to allow the transmitter to broadcast a status message at power on.

Table 67: Sample Sequence 3

esav ANN=TONE 1.0
esav ANN=CWPM 20,-1,-1,-1,-1
esav ANN=BEGN
esav ANN=BATC V 7.2
esav ANN=BATC EV 7.2
esav ANN=DONE
esav ANN=FREQ 144.285
esav ANN=TONE 1.6
esav ANN=CWPM 15
esav ANN=STAT

9.4.1 TONE

Set the pitch of the code.

The pitch of the dits and dahs may be useful as an indication of what type of message this is.

9.4.2 CWPM

Set the message to a comfortably high speed to get the message sent quickly.

9.4.3 BEGN

Send the *CQ* message with our callsign.

9.4.4 BATC

Send battery report in code (as opposed to voice).

Both forms report battery voltage with the second encoding the volts and tenths into a seried of TTTT and EEE characters. Number of /textbfdah characters representing volts and the number of /textbfdit characters representing tenths.

9.4.5 DONE

Send our callsign and the *SK* message.

9.4.6 FREQ

Change over to the operating frequency for the group.

9.4.7 TONE

Change over to the target audio frequency.

9.4.8 CWPM

Change over to the target chipping rate.

9.4.9 STAT

This sends text to the UART.

This is a diagnostic aid for when you are hooked up to the USB port to configure the device.

9.5 Sample Sequences

Example code for setting up a FOX Transmitter.

9.5.1 Schedule 1 Program

This code is the *S1* sequence.

Table 68: Sample Sequence 4

esav S1=CWPM 15
esav S1=BEGN
esav S1=WAIT 1
esav S1=CODE IOWA CITY
esav S1=CODE AMATEUR RADIO
esav S1=CODE CLUB FOX HUNT
esav S1=WAIT 1
esav S1=CODE This is a test
esav S1=CODE of the code
esav S1=CODE generator in the
esav S1=CODE KC0JFQ FOX TX
esav S1=CODE built for the
esav S1=CODE IOWA CITY
esav S1=CODE AMATEUR RADIO
esav S1=CODE CLUB FOX HUNT
esav S1=WAIT 3
esav S1=BATC V
esav S1=WAIT 3
esav S1=DONE

9.5.2 CWPM

In this example, we change the chipping rate for the CW generator at the start of each sequence. The chipping rate can be changed at any time, i.e. in the middle of a message, if desired.

9.5.3 BEGN

Beginning of message marker. This enables the RF section and sends out a CQ call with the stored callsign.
The current system time is stored when this command runs so it can be differenced with the current system time at a later point to measure the time required to send a message.

9.5.4 WAIT

Leaves an unmodulated carrier on and waits the specified number of seconds.

9.5.5 CODE

This sends a few characters of the message. Each storage record holds up to 25 bytes of message text.

9.5.6 DONE

Terminates the message. Sends the callsign and *SK* to end the message. If the USB UART is connected, the time required to send the message is in the status message.

10 Practical Sequencing

An exercise in maintaining your sanity.

10.1 Programs

Programs that are used to initialize and announce.

The handling of the **TEST** and **MAS** jumpers was reorganized in the V3.54 release. The jumpers now define one of four operating *modes*.

Three of the modes appear very similar to the previous release. Added in the V3.54 release is an error recovery mode where both jumpers may be installed to completely suppress processing commands during startup. This leaves the system completely unconfigured allowing the FRAM to be erased and reloaded.

See the table in section 4.3.1 on page 48.

10.1.1 INI=

Basic Initialization. This separates station specific setup tasks from the operating sequence.

This is the station specific setup, unique to each foxhunt transmitting station.

```
INI=NAME,xxx                "tactical call"
INI=CALL,xxx                "station callsign"
INI=EPOC,-5.0               "CDT, offset to local time from ZULU"
INI=TIME                    "get time from DS1672"
INI=CONF,SI5351             "Hardware we’re running on"
INI=CONF,CLK2,8MA           "SI5351 output configuration"
INI=FREQ,144.150            "Common announcement frequency"
INI=REM-,FOX_INIT_2023.FOX "comment with source filename"
INI=REM-,STRT,00:00:00      "comment"
INI=MODS,S0,300,0           "Schedule 0"
INI=MODS,S1,300,60          "Schedule 1"
INI=MODS,S2,300,120         "Schedule 2"
INI=MODS,S3,300,180         "Schedule 3"
INI=MODS,S4,300,240         "Schedule 4"
```

10.1.2 TEST=

Test *sequence*. This *sequence* runs when the **TEST** jumper is installed and the **MAS** jumper is not installed.

As the **INI=** commands have already run, we have loaded the callsign and nickname into the running system as well as the current time from the TOY clock.
The **TEST=** commands may change or redefine the configuration as needed.

```
TEST=CONF,SI5351           "Same as INI="
TEST=CONF,CLK0,8MA         "Testing the other RF path"
TEST=FREQ,144.150          "Same as INI="
TEST=CWPM,30,-1,-1,-1,-1   "Fast code so we can manually drive the system"
TEST=CONF                  "Status Report"
TEST=STAT                  "Status Report"
```

10.1.3 MAS=

The **MAS=** *sequence* will run when the **MAS** jumper is installed and the **TEST** jumper is not installed..

It is run after the **INI=** *sequence.*

```
MAS=CWPM 35,-1,-1,-1,-1
MAS=STAT
```

10.1.4 ANN=

Announcement message.
This sequence is run after the **INI=** sequence when neither the **TEST** jumper nor the **MAS** jumper are installed.

The station common setup was already done in the **INI=** sequence, so it's not repeated.

This message tells the hunt organizer that this station is alive and transmitting when it is turned on.

Note the parameter substitution for the callsign and tactical callsign.

```
ANN=REM-,FOX_ANN_V2023.FOX  "comment with source filename"
ANN=TONE,1.0                "audio frequency, KHz"
ANN=CWPM,20,-1,-1,-1,-1     "Code rate"                                9574
ANN=BEGN                    "Begin transmitting (RF on)"
ANN=TALK,<CALL>             "verbal callsign"
ANN=TALK,<NAME>             "verbal tactical call"
ANN=WAIT,1                  "delay 1 second"
ANN=BATV,V                  "verbal battery report, Voltage"
ANN=BATV,I                  "verbal battery report, Current"
ANN=WAIT,.3                 "delay 300mS"
ANN=DONE                    "Done transmitting (RF off)"
ANN=FREQ,144.250            "Change to new operating frequency"
ANN=STAT                    "status report (in case you're looking)"
ANN=RUNO,SO                 "Enable the ZERO schedule"
ANN=REM-                    "Remark"
```

10.1.5 S0=

An example message block. This is sent on the *S0* schedule.

The station reports, in this example, are sent with an audio tone frequency of 1KHz at 20WPM. The other parameters to the CWPM command select standard timing.

The body of the message is sent at a different audio frequency and a different CW rate.

```
S0=REM-,1,MINUTE,25WPM      "Comment text"
S0=CWPM,20,-1,-1,-1,-1      "Set code to 20WPM with standard timing"
S0=TONE,1.0                 "Set audio tone"
S0=BEGN                     "Begin transmitting (RF on)"
S0=TALK,<CALL>              "verbal callsign"
S0=TALK,<NAME>              "verbal tactical call"
S0=WAIT,0.5                 "500mS delay, separate CW from voice"
S0=BATV,V                   "verbal battery report, Voltage"
S0=BATV,I                   "verbal battery report, Current"
S0=WAIT,0.5                 "500mS delay, separate voice from CW"
S0=TONE,1.2                 "Change audio tone"
S0=CWPM,25,-1,-1,-1,-1      "change code to 25WPM"
S0=WAIT,0.5                 "500mS delay (not useful here)"
S0=CODE,IOWA,CITY           "send CW message"
S0=CODE,AMATEUR,RADIO       "more"
S0=WAIT,0.5                 "500mS delay"
S0=CWPM,20,-1,-1,-1,-1      "change code back to 20WPM"
S0=TONE,1.0                 "set audio tone back"
S0=DONE                     "Done transmitting (RF off)"
```

9620

Most of the 60 second message is taken sending station status (i.e. the battery condition) and the basic FCC identification.

S0=BEGN sends a CQ with the callsign. We also verbalize the battery condition to allow the event host to monitor station operation.

DONE sends callsign.followed by SK to comply with FCC identification requirements.

10.1.6 S0= (102-73161 circuit board)

An alternate message block for the 102-73161-25 boards. This is sent on the *S0* schedule.

Similar to the previous schedule, but tailored to the smaller memory footprint with the 102-73161 boards.

The callsign and tactical callsign are stored in the back of FRAM, because the FRAM is much smaller than the FLASH device on the 102-73181 boards. We can identify the unit verbally without having to be able to read code.

The battery report is not verbal, but in code.
The **BATC,V,7.2** request a voltage report with a low battery limit of 7.2 volts. If the battery voltage is above the specified trip point, the code report will start with *BATC HI HI* and if below the trip point the report starts with *BATC SOS SOS*. These two patterns are easily recognized to allow the battery voltage to be picked out of the code stream.

The **BATC,EV,7.2** switch from sending the battery voltage directly in code to sending an encoded form, again being easy to read for those not proficient in code. The units voltage is sent as a series of **T** (*dah*) characters. The tenths field is sent a a series of **E** (*dit*) characters.

```
S0=CWPM,20,-1,-1,-1,-1     "Set code to 20WPM with standard timing"
S0=TONE,1.0                "Set audio tone"
S0=BEGN                    "Begin transmitting (RF on)"
S0=TALK,<CALL>             "verbal callsign"
S0=TALK,<NAME>             "verbal tactical call"
S0=WAIT,0.5                "500mS delay, separate CW from voice"
S0=BATC,V,7.2              "CW battery voltage report, reading in code"
S0=BATC,EV,7.2             "CW battery voltage report, reading simple encoding"
S0=WAIT,0.5                "500mS delay, separate voice from CW"
S0=TONE,1.2                "Change audio tone"
S0=CWPM,25,-1,-1,-1,-1     "change code to 25WPM"
S0=WAIT,0.5                "500mS delay (not useful here)"
S0=CODE,IOWA,CITY          "send CW message"
S0=CODE,AMATEUR,RADIO      "more"
S0=WAIT,0.5                "500mS delay"
S0=CWPM,20,-1,-1,-1,-1     "change code back to 20WPM"
S0=TONE,1.0                "set audio tone back"
S0=DONE                    "Done transmitting (RF off)"
```

10.2 Managing Schedules

Assume that you have the schedules stored on a host of some type, nominally a Linux box as that's where the download management utility was created.

Create a file for the master unit, for example *Fox_1.fox*. Place the initialization commands in this file. In particular the unique callsign for the first unit, something like *KA0ABC/1*, and the operating frequency, such as *esav INI=FREQ 144.299*. Look in the examples for a more meaningful list.

We will also define all of the schedules in this file, these are the *esav INI=MODS S1 10:00 00:30* lines that define when each message sequence will be transmitted.

Set the master time update command: *esav MAS=TSND*

Set the master: *esav INI=MODS MAS 5 3*.

All the messages are scheduled to begin at the very beginning of the period. This staggering of the time update command prevents the time update message from causing the message traffic to be dropped.

Create your message traffic, one message group to a file, using a sequential schedule numbers for each. The first file would be, fox example, *message_1.fox* followed by *message_2.fox*, etc.

These should define the audio tone *sav S1=TONE 1.4*, the chipping rate *esav S1=CWPM 20,-1,-1,-1,-1*, and initiate message transmission *esav S1=BEGN*.

The text of the message follows, a few words per line as follows: *esav S1=CODE FOX HUNT*. Keep in mind that each record in the flash file system is limited to 32 characters, leaving less than 28 available for actual message text.

End the transmission with *esav S1=DONE* and *esav S1=TSET* to power down the transmitter and update the system clock from the TOY clock.

Now you can go back and add *#include message_1.fox* for each message file to each of the *Fox_1.fox* files to include the messages. Given a large FRAM device, this will all fit.

Now you can used the *fox_simple* utility to download the *Fox_1.fox* file into the first fox. The *#include* lines will drag in the message text. Each unit will have an identical schedule and message buffers.

At this point you can manually remove the unwanted schedules from each unit using the *ERAS n* command. It is vital the the *ERAS n* command is used to clear the unwanted schedule commands from memory as the *EZER* command will zero out the record, hiding the rest of the setup information in the FRAM from view.

To replace a schedule, we would use the *EZER* command to clear the command to be replaced and then use *ESAV* to load the replacement command which will end up being placed into the zero-ed command and making everything visible again.

10.3 Time Synchronization days(s) prior to foxhunt

The following may all be checked days before the fox hunt.

Remember to bring the 3.5mm stereo patch cable along to connect the units together.

Make sure the *HDX* jumper is in place at least on the master board, but preferably on both sides. This eliminates the need to have tip and ring swapped in the patch cable.

Make sure the *MAS* jumper is in place on the master unit.

Verify that the FRAM is loaded and schedules are enabled as intended.

10.4 Time Synchronization day of the foxhunt

To synchronize time before the fox hunt, connect the synchronization cable up to the master.

High power units may require a 50 Ohm termination when no antenna is connected. Units without an amplifier may be operated without an antenna without causing problems.

One by one connect each slave to the master and power up the slave. Turn the master on for about 15 to 20 seconds to allow it to send the time update message. Power off **both** units, this keeps the master from sending a message which suppresses sending the time message.

The units may be left off until they are in place to conserve battery power.

10.5 Audio Frequency

The system is designed to allow operating multiple units on a single frequency while having only one transmitter active at a time. To make distinguishing individual units a bit simpler, each unit may be configured to produce a unique audio frequency. the *TONE* command may be added at the beginning of the message sequence.

This feature may also be used to allow *masquerading*. Consider a hunt with three known or advertised transmitters, each of which operates on a unique audio tone. A fourth unit can *masquerade* and any (or all) of the known/advertised units.

The tone may also change within a message. Simply adding a tone command will change the tone at the point the command is encountered.

10.6 Carrier Frequency

The system is also designed to allow selecting the carrier frequency.

Nominally, you would select a carrier frequency in the initialization file and always operate on this frequency. This is simply a reasonable way to operate, the software does not impose any limit on selecting frequency.

The frequency is set by command and that command may occur as many times as desired..

Each message may be sent on a different frequency or the frequency may change in the middle of a message.

10.7 Accessing the USB port

The USB port is inaccessible inside the box on older boards, requiring unscrewing and removing the cover.

Having an exposed USB port in the field is inviting foreign material to take up residence in the exposed connector. Is is **not** recommended to modify the case to allow use of the USB connector with the cover in place. The unit will operate on the bench with the cover removed.

The 102-73181-10 units move the the serial port over to the connector was the network time port to providfe access without having to open the box.

The network time port was never used, it having been reallocated to the DA818/DRA818 tranceiver modules.

11 Audio File Utility

The audio utility program, *pwm_audio_util*, is used to gather all of the audio clips that are to be loaded into the fox transmitter together into a single Intel HEX file that can be loaded into the fox transmitter FRAM.

The fox transmitter recognizes the record format on an Intel HEX file. When an Intel HEX arrives through the command port (i.e. the USB port) it is decoded and saved to the FRAM.

The audio utility program allocates space starting at the top of FRAM and works toward the bottom of the device where the *Configuration Commands* are located. As the audio utility program is independent of the load utility, space allocation is manual. You must insure that the audio file system does not overlap with the *Configuration Command File System.*

11.1 Clip File

This file is s simple flat file listing all of the raw files that are to be gathered together into an audio file for download into the transmitters FRAM.

11.2 Output File

The output of the *Audio File Utility* is an Intel HEX file along with updated command-line arguments for use with the next invocation of the *Audio File Utility.*

Although there are flags to eliminate the RIFF/WAVE header from the input wave file, the **V3.27** revision of the fox transmitter software uses the RIFF/WAVE header to determine the sample count and sample rate of the audio clip.

11.3 Downloading the audio image

Downloading into the fox station is all plain text. The download utilities simply streamline the process of feeding command lists and hex files to the target fox station.

Progress can be monitored using the *halo_term* utility, it will share the USB connection with the *fox_simple* utility. The *halo_term* screen will display the status response from the target fox station as *fox_simple* sends hex records.

The compiled Intel HEX file may be directly loaded into the waveform memory (FLASH) using the *fox_simple* utility:

```
        /home/wtr/Radio/halo_term/fox_simple \
                -STACH \
                -c100 \
                -t10 \
                -f/home/wtr/WAV/fox_73181_r4k.hex
```

-S flag

This selects the USB port through which the download operation will occur. A list of keywords may be found in the *halo_term* utility by using the "-H" flag.

-c flag

Sets the inter-line delay (in milliseconds). This is used to speed up the download, although it is still painfully slow.

A 10mS delay works on all the fox transmitters produced by the author.z

-t flag

Time truncation flag selects the number of days the timetag is truncated to. This simply reduces the size of the time field sent to the target at the start of the download.

-f flag

File selection flag. This will be the Intel HEX file that is to be downladed.

There are directory records embedded in the Intel HEX file produced by the audio utility, but they are prefaced by a *REMARK* command so they aren't loaded into FRAM.

Use *grep* to extract the TALK records:

```
grep TALK fox_73181_r4k.hex
```

You end up with a list similar to this:

```
REM- esav TALK=V_HZ 15232
REM- esav TALK=V_KHZ 17664
REM- esav TALK=V_MHZ 20864
REM- esav TALK=V_N0 24064
REM- esav TALK=V_N1 26752
REM- esav TALK=V_N2 28544
REM- esav TALK=V_N3 30720
REM- esav TALK=V_N4 32640
REM- esav TALK=V_N5 34560
REM- esav TALK=V_N6 36736
REM- esav TALK=V_N7 38528
REM- esav TALK=V_N8 40448
REM- esav TALK=V_N9 41984
```

Use your editor to remove the **REM-** from each line and insert the TALK Directory into the appropriate file or simply download it using the *fox_simple* utility.

12 FOX Transmitter Label Utility

This is a utility to generate labels, cards, and log sheets for the ICARC fox junt.

A list of the transmitters in the *transmitter stable* is stored in the *fox_label.csv* file. These are all the transmitters that are functional. We may not need all of the available transmitters.

The **fox_label** program is run once to produce all of the files that may need to be printed for the fox hunt. All files are produce in order to keep the **ID** and **validation code** consistent. The codes are randomly generated every time the **fox_label** program runs.

12.1 fox_label program

This is the utility that produces all the printable documebnts for the hunt. Running it produces the following:

Table 69: fox_label output files

	Type	**Contents**	**Description**
1	label	fox_label_A_top.ps	Transmitter FCC Identification
2	label	fox_label_A_bot.ps	Fox Hunt Identification
3	sheet	fox_label_A_chk.ps	Fox Hunt Setup Checklist
4	card	fox_label_B_hunter.ps	Fox Hunt Participant Card
5	card	fox_label_C_FOX0.ps	Fox Hunt Transmitter Found Card
n	card	fox_label_C_FOXn.ps	(14 of them in total)

When the **fox_label** program runs, it deletes all of the *FOX Transmitter Found Cards* files before it produces new ones. This behavior is intended to eliminate stale files when you are setting up for the next hunt. If you're not paying close attention when you ask for new *Found Cards*, old ones left from the last run cound be used inadvertantly. Using the wrong offset or changing the contents of the *fox_label.csv* file could create this hazard.

Eliminating the old ones attempts to eliminate this hazard.

Example of how the author uses the program. Refer to the csv file used in this example, *fox_label.csv* in section 12.7 on page 165.

First we print the pages to manage the hunt beforehand and at the checkin station.

```
./fox_label -CW0JV -e"FW Kent Park FOX HUNT" -o8
lp fox_label_A_chk.ps           <--- plain paper
lp fox_label_A_bot.ps           <--- 14-up label stock
ls -l fox_label_A_bot.ps        <--- we do these only once!
lp -n2 fox_label_B_hunter.ps    <--- 10-up business cards
```

Now print the *Found Cards* for the new transmitters. For the ICARC hunt, these are the transmitters that are spread out across the park (wide area).

```
./fox_label -CW0JV -e"FW Kent Park FOX HUNT" -o8
lp fox_label_C_FOX21.ps         <--- 10-up business cards
lp fox_label_C_FOX22.ps         <--- 10-up business cards
lp fox_label_C_FOX23.ps         <--- 10-up business cards
lp fox_label_C_FOX24.ps         <--- 10-up business cards
lp fox_label_C_FOX25.ps         <--- 10-up business cards
lp fox_label_C_FOX26.ps         <--- 10-up business cards
```

Now print the *Found Cards* for the old transmitters. For the ICARC hunt, these are the transmitters that are close-in, within walking distance.

```
./fox_label -CW0JV -e"FW Kent Park FOX HUNT" -o8
lp fox_label_C_FOX5.ps          <--- 10-up business cards
lp fox_label_C_FOX6.ps          <--- 10-up business cards
lp fox_label_C_FOX7.ps          <--- 10-up business cards
lp fox_label_C_FOX8.ps          <--- 10-up business cards
lp fox_label_C_FOX20.ps         <--- 10-up business cards
```

12.1.1 fox_label -A

Avery_Number.

5329
Half size business cards. More per sheet, so less cost?
1 by 3 inch cards, 16 up.

5371
Full size business cards. Get 'em cheap on Amazon.
2 x 3 1/2 inch cards, 10 up.

12.1.2 fox_label -C

SECONDARY_Callsign.

Sets the callsign that appears on labels, cards and log sheets.

12.1.3 fox_label -o

O_offset.

Entry offset in the *fox_label.csv* file.
Comment lines are not counted.
Using -o0 starts at the the first entry.

12.1.4 fox_label -d

DBG_Level.

Increases the debug level by one.
Currently level 1 is everything!

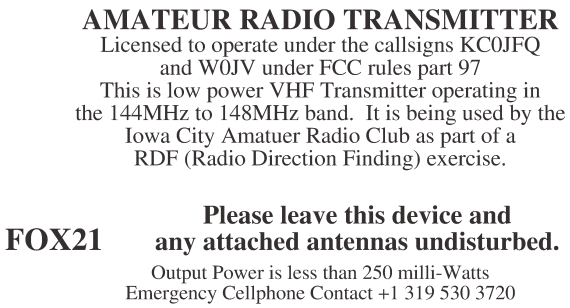
12.1.5 fox_label -e

Event Name.

Sets the event name that appears on labels, cards and log sheets.

12.2 fox_label_A_top

These labels are provided to permanantly attach to the transmitter to identify it and provide contact information in the event it is lost or inadvertantly removed.



AMATEUR RADIO TRANSMITTER
Licensed to operate under the callsigns KC0JFQ
and W0JV under FCC rules part 97
This is low power VHF Transmitter operating in
the 144MHz to 148MHz band. It is being used by the
Iowa City Amatuer Radio Club as part of a
RDF (Radio Direction Finding) exercise.

FOX21

Please leave this device and
any attached antennas undisturbed.
Output Power is less than 250 milli-Watts
Emergency Cellphone Contact +1 319 530 3720

Figure 24: FOX Transmitter TOP Labels

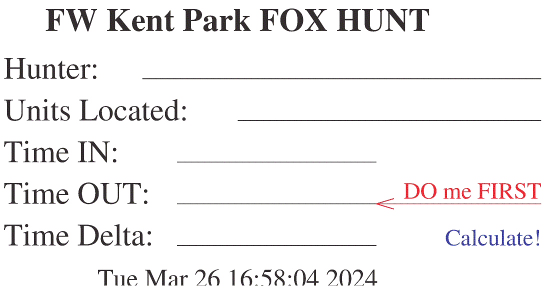
Note that these labels are labeled with the transmitter nickname. They should be attached to the side of the enclosure that the circuit board is mounted when the fox transmitter is first put into service.

These labels are not keyed to a particular hunt.

Use the *-o <n>* argument to **fox_label** to select the starting point in the csv file.

12.3 FOX HUNT Checkin Card

The *Hunter Cards* can be printed as a convenient means of keeping track of the participants.



FW Kent Park FOX HUNT

Hunter: ____________________

Units Located: ____________________

Time IN: ____________________

Time OUT: ____________________ DO me FIRST

Time Delta: ____________________ Calculate!

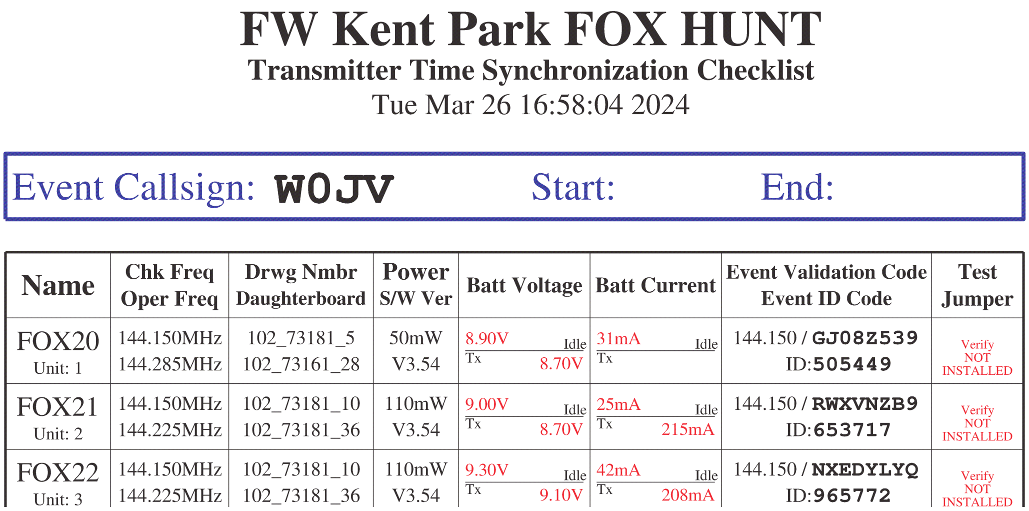
Tue Mar 26 16:58:04 2024

Figure 25: FOX HUNT Checkin Card

Noting that the only time sensitive information on the card is the date the cards were generated, it is not really necessary to wait until the last minute to produce these.

12.4 FOX HUNT Check Sheet

This is the check sheet for the organizer. Use it for collecting together the transmitters required for the hunt. There are open areas on the sheet to record battery voltage and current observed when the TOY clock is set the day before the hunt. The **fox_label_A_bot** file must be printed from the same run!



FW Kent Park FOX HUNT

Transmitter Time Synchronization Checklist

Tue Mar 26 16:58:04 2024

Event Callsign: **W0JV** Start: End:

10222

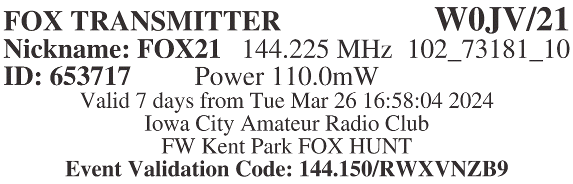
Name	Chk Freq Oper Freq	Drwg Nmbr Daughterboard	Power S/W Ver	Batt Voltage	Batt Current	Event Validation Code Event ID Code	Test Jumper
FOX20 Unit: 1	144.150MHz 144.285MHz	102_73181_5 102_73161_28	50mW V3.54	8.90V Idle Tx 8.70V	31mA Idle Tx	144.150 / **GJ08Z539** ID:**505449**	Verify NOT INSTALLED
FOX21 Unit: 2	144.150MHz 144.225MHz	102_73181_10 102_73181_36	110mW V3.54	9.00V Idle Tx 8.70V	25mA Idle Tx 215mA	144.150 / **RWXVNZB9** ID:**653717**	Verify NOT INSTALLED
FOX22 Unit: 3	144.150MHz 144.225MHz	102_73181_10 102_73181_36	110mW V3.54	9.30V Idle Tx 9.10V	42mA Idle Tx 208mA	144.150 / **NXEDYLYQ** ID:**965772**	Verify NOT INSTALLED

Figure 26: FOX HUNT Check Sheet

10227

12.5 fox_label_A_bot

These labels are unique to the fox hunt. The labels have a generated **ID** and **validation code**. They should be produced, printed, and affixed the day before the hunt. The **fox_label_A_chk** must be printed from the same run!



FOX TRANSMITTER **W0JV/21**
Nickname: FOX21 144.225 MHz 102_73181_10
10244 **ID: 653717** Power 110.0mW
Valid 7 days from Tue Mar 26 16:58:04 2024
Iowa City Amateur Radio Club
FW Kent Park FOX HUNT
Event Validation Code: 144.150/RWXVNZB9

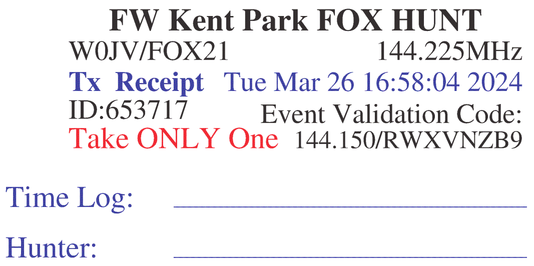
Figure 27: FOX Transmitter BOT Labels

These labesl are placed on the other half of the enclosure. These labels are generated before each hunt and affixed to the trasmitter. The labels have ID values that generated when the label utility runs. 10252

The **fox_label_A_bot** and the **fox_label_A_chk** files should be generated in one run of the **fox_label** program to keep the **ID** and **validation code** values synchronized. 10260

12.6 fox_label_C_FOX*

These labels are printed on cardstock to be left with the transmitter to be picked up by the hunters. The **fox_label_A_bot** and **fox_label_A_chk** must be printed from the same run!



FW Kent Park FOX HUNT
W0JV/FOX21 144.225MHz
Tx Receipt Tue Mar 26 16:58:04 2024
ID:653717 Event Validation Code:
Take ONLY One 144.150/RWXVNZB9 10275

Time Log: ______________________

Hunter: ______________________

Figure 28: FOX Transmitter Found Cards

The *Found Cards* are printed on business card stock, 10 per page. The **fox_label** utility produces files for each of the transmitters listed in the **fox_label_A_bot** file (14 of them). 10283

Simpply print a card for each transmitter, split them out and rubberband them to their transmitter. Hunters then take a card from the transmitter to document that they did, in fact, find the transmitter. 10292

12.7 fox_label.csv

The transmitter information file.

Listing 8: fox_label.csv

```
 21,  KC0JFQ,  144.225, 144.150, V3.62, 102_73181_10/102_73181_36, 20.000, 110.0,  9.0, 8.7,  25, 215
 22,  KC0JFQ,  144.225, 144.150, V3.62, 102_73181_10/102_73181_36, 20.000, 110.0,  9.3, 9.1,  42, 208
 23,  KC0JFQ,  144.225, 144.150, V3.62, 102_73181_10/102_73181_36, 20.000, 140.0,  8.9, 8.5,  56, 338
 24,  KC0JFQ,  144.225, 144.150, V3.62, 102_73181_10/102_73181_36, 20.000, 100.0,  9.2, 9.0,  51, 310
 25,  KC0JFQ,  144.225, 144.150, V3.62, 102_73181_10/102_73181_36, 20.000, 120.0,  9.0, 8.6,  42, 317
 26,  KC0JFQ,  144.225, 144.150, V3.62, 102_73181_10/102_73181_36, 20.000, 140.0,  8.1, 7.9,  48, 337
#
 27,  KC0JFQ,  144.325, 144.150, V3.62, 102_73181_10/102_73161_28, 20.000,  50.0,  9.0, 8.9,  24, 133
 28,  KC0JFQ,  144.325, 144.150, V3.62, 102_73181_10/102_73161_28, 20.000,  50.0,  9.0, 8.9,  24, 133
 29,  KC0JFQ,  144.325, 144.150, V3.62, 102_73181_10/102_73161_28, 20.000,  50.0,  9.0, 8.9,  24, 133
 30,  KC0JFQ,  144.325, 144.150, V3.62, 102_73181_10/102_73161_28, 20.000,  50.0,  9.0, 8.9,  24, 133
 31,  KC0JFQ,  144.325, 144.150, V3.62, 102_73181_10/102_73161_28, 20.000,  50.0,  9.0, 8.9,  24, 133
 32,  KC0JFQ,  144.325, 144.150, V3.62, 102_73181_10/102_73161_28, 20.000,  50.0,  9.0, 8.9,  24, 133
#
 20,  KC0JFQ,  144.285, 144.150, V3.62, 102_73181_5/102_73161_28,  20.000,  50.0,  8.9, 8.7,  31,   0
  8,  KC0JFQ,  144.285, 144.150, V3.56, 102_73161_25/102_73161_24, 20.000,   5.0,  8.4, 8.3,   0,   0
  7,  KC0JFQ,  144.285, 144.150, V3.56, 102_73161_25/102_73161_24, 20.000,   5.0,  8.7, 8.6,   0,   0
  6,  KC0JFQ,  144.285, 144.150, V3.56, 102_73161_25/102_73161_24, 20.000,  45.0,  8.1, 7.9,   0,   0
  5,  KC0JFQ,  144.285, 144.150, V3.56, 102_73161_25/102_73161_24, 20.000,  45.0,  8.3, 8.2,   0,   0
#
  2,  KC0JFQ,  144.305, 144.305, V2.03, 102_73161_12,              24.576,  12.0,  0.0, 0.0,   0,   0
  3,  KC0JFQ,  144.335, 144.335, V1.52, 102_73161_12,              24.576,  12.0,  0.0, 0.0,   0,   0
  4,  KC0JFQ,  144.565, 144.565, V1.52, 102_73161_12,              24.576,   5.0,  0.0, 0.0,   0,   0
#
#
```

When calling **fox_label** with the **-o** argument, the numberic value to **-o** is the number of text lines in the csv file to skip. Our **-o 8** used above skips ove the first 8 lines.

12.7.1 fox_label.csv; Column 1

FOX Unit Number.

This number is somewhat arbitrary. Units are just numbered in sequential order. Breaks in the sequence are allowed.

12.7.2 fox_label.csv; Column 2

Owner Callsign.

This is the callsign that appears in labesl **only** when a callsign (using the **-C** flag) is not specified.

12.7.3 fox_label.csv; Column 3

Operating Frequency.

You will operate the hunt on this frequency.

12.7.4 fox_label.csv; Column 4

Setup Frequency.

The transmitter sends setup confirmation and status on this frequency when switched on.

A properly configured system will switch over to the operating frequency after the setup message has been sent.

12.7.5 fox_label.csv; Column 5

zNEO software revision.

The software revision of the operating software.

12.7.6 fox_label.csv; Column 6

Transmitter hardware revision and daughterboard hardware revision.

These are the drawing numbers for the main transmitter board and the attached RF daughterboard.

12.7.7 fox_label.csv; Column 7

Reference crystal on the RF synthesizer.

This is the crystal used by the ICS525/ICS307/SI5351.

12.7.8 fox_label.csv; Column 8

Measured Transmit Power.

Measured during initial testing. Essentially with the transmitter on the bench, connected to a power meter, when it is first configured and mated with an RF daughterboard.

All of the RF power is flowing to the meter so the current sense circuit works correctly and that measurement probably comes as a byproduct of this measurement.

12.7.9 fox_label.csv; Column 9

Idle Battery Voltage.

Battery voltage reported by the transmitter when not transmitting.

This voltage can be seen when the transmitter is connected to a host system through the serial cable using the **STAT** command.

12.7.10 fox_label.csv; Column 10

Transmitting battery voltage.

Battery voltage reported by the transmitter when transmitting.

This voltage shows up in battery reporting commands when transmitting.

This voltage can be seen when the transmitter is connected to a host system through the serial cable using the **STAT** command. Both idle voltage and transmit voltage are visible when using the **STAT** command.

12.7.11 fox_label.csv; Column 11

Idle Battery Current.

Battery current reported by the transmitter when not transmitting.

This current can be seen when the transmitter is connected to a host system through the serial cable using the **STAT** command.

12.7.12 fox_label.csv; Column 12

Transmitting Battery Current

Battery current reported by the transmitter when transmitting.

This current shows up in battery reporting commands when transmitting. Some units seem to have difficulty with this when using higher powered RF amplifiers.

This current can be seen when the transmitter is connected to a host system through the serial cable using the **STAT** command. Both idle current and transmit current are visible when using the **STAT** command.

13 Synthesizer configuration utilities

These are the utilities used to build the configuration tables for the RF synthesizer chips.

13.1 SI5351 configuration table utility

This is a bit of code pulled together from various sources that is used to build the configuration table for the SI5351 device.
Nominally, the tables are stored in the zNEO program flash for immediate use by the **FREQ** command.
Table space in the zNEO is limited so there are a limited number of entries in this table.

The internal table is initially produced by the *si5351a_calc* utility and compiled into the si5351 driver. The contents of the table may be dumped using the command: **5351 TABLE**. If a frequency you wish to make use of is not in the table, the *si5351a_calc* utility may be used to generate the multisynth parameters necessary to configure the si5351 for that frequency.

The zNEO software suite may then be re-compiled (assuming you have the requisite tools) or loaded using the **5351 FREQ** *frequency*, **5351 PLLS** *P1, P2, P3* and **5351 MS** *P1, P2, P3* commands to store the setup values.
These commands may be placed into the INI= file or the S*n*= file for field use.

The *si5351a_calc* utility builds a c fragment like the following:

```
//
//  Version V1.0
//
//  ./si5351a_calc -T2 -F 144.1,144.21,25.0 -o -17
//
//     Using a crystal frequency of: 20.000 MHz
//     Frequency Offset   -17.000KHz
//     Starting Frequency 144.100MHz
//     Ending   Frequency 144.210MHz
//     Frequency Step     25.000KHz
//     Calculation method TWO  kc0jfq_setup()
//
rom float  SI5351_CRYSTAL = 20.000;
rom char   si5351_calc[] = {"./si5351a_calc -T2 -F 144.1,144.21,25.0 -o -17 "};
//
rom struct SI5351_FREQ_TBLE si5351_frequency_table[] = {
//                 Stage 1 Synthesizer    Stage 2 Synthesizer       Internal      Output
//      char*      long     long     long     int    int    int          VFO      Multisynth
//    Frequency   MSNxP1   MSNxP2   MSNxP3  MSx_P1 MSx_P2 MSx_P3      Frequency     Divisor
    {R"144.100", 0x139C, 0xC0300, 0xF4240, 0x0100, 0x00,  0x01  },  // 864.498   1    6   T2 OFF=-17.0K
    {R"144.125", 0x139D, 0xB66BF, 0xF4240, 0x0100, 0x00,  0x01  },  // 864.648   2    6   T2 OFF=-17.0K
    {R"144.150", 0x139E, 0xACA7F, 0xF4240, 0x0100, 0x00,  0x01  },  // 864.798   3    6   T2 OFF=-17.0K
    {R"144.175", 0x139F, 0xA2E3F, 0xF4240, 0x0100, 0x00,  0x01  },  // 864.948   4    6   T2 OFF=-17.0K
    {R"144.200", 0x13A0, 0x991FF, 0xF4240, 0x0100, 0x00,  0x01  },  // 865.098   5    6   T2 OFF=-17.0K
    {      NULL,       0,      0,       0,      0,    0,     0  } };
```

The comment text indicates the parameters used to generate the file; the frequency range, the frequency step size and the crystal used by the SI5351 on the circuit board.

There is a provision for the application of a frequency offset as part of a debugging effort.

Commands used to run a quick test of the 144.175MHz MSNA divisors follows:

```
5351 TEST 139F,A2E3F,F4240
```

This will load the SI5351 and generate a stream of *HI HI HI...* until a keypress.
If access to the MSA multisynth is required, it can be loaded as well:

```
5351 FREQ 144.175,0
5351 PLLS 139F,A2E3F,F4240
5351 MS 100,0,1
```

The path through the SI5351 can be verified by loading them into the SI5351 and then dumping the SI5351 registers to see that the patterns were correctly formed and loaded.

```
5351 LOAD
5351 DUMP
```

The **5351 LOAD** command takes the patterns stored using the **FREQ**, **PLLS**, and **MS** subcommands and sends them to the SI5351.

Then the SI5351 register dump (**5351 DUMP**) verifies that the patterns were correctly processed and loaded.

13.1.1 Frequency Tuning

The *SI5351 configuration table utility* can be used to zero in on the best divisors using only a handie-talkie by generating a table around the target frequency. The MSNA divisors can be loaded to find the set that best centers on the target frequency.

The example here is a test generated during testing of the prototype unit. A frequency offset of 17KHz appears in the system (this is corrected using the **-o -17** controls).

```
./si5351a_calc -C2 -F 144.14,144.16,1.0 -o -17 | grep TEST > si5351a_test_zero.c
```

```
            5351 TEST 139E,4EE7F,F4240
            5351 TEST 139E,58480,F4240
            5351 TEST 139E,61A7F,F4240
            5351 TEST 139E,6B080,F4240
            5351 TEST 139E,7467F,F4240
            5351 TEST 139E,7DC7F,F4240
            5351 TEST 139E,87280,F4240
            5351 TEST 139E,90880,F4240
            5351 TEST 139E,99E80,F4240
            5351 TEST 139E,A347F,F4240
            5351 TEST 139E,ACA7F,F4240
            5351 TEST 139E,B6080,F4240
            5351 TEST 139E,BF680,F4240
            5351 TEST 139E,C8C80,F4240
            5351 TEST 139E,D2280,F4240
            5351 TEST 139E,DB87F,F4240
            5351 TEST 139E,E4E80,F4240
            5351 TEST 139E,EE47F,F4240
            5351 TEST 139F,03840,F4240
            5351 TEST 139F,0CE3F,F4240
            5351 TEST 139F,16440,F4240
```

The commands can be copied into the fox transmitter and the handie-talkie can move one step in either direction to gauge that the SI5351 is generating the correct frequency.

A frequency counter would also be of use here. Use a 102-73161-22 **AMP BYPASS** board to connect the output of the SI5351 directly to the BNC connector.

13.1.2 Synthesis Divisor calculation method TWO

You should notice that the table fragment earlier indicates the method used to calculate the divisors. The code for **method TWO** is detailed here.

Listing 9: si5351a_calc-172

```
int kc0jfq_setup(double xtal, double fout, int *ms_values, double *abcdef, int flag){
    double bc128_floor;
    double def_floor;
    double abc_fvco;
    double MSNx_P[4] = {4*0.0};
    double MSx_P[4] = {4*0.0};
    double fa;
    int sts;
```

What we need to build the divisor table:

1. **xtal**

   This is the SI5351 crystal frequency, X5.

2. **fout**

   This is the output frequency that will appear on one of the CLKn pins.

3. **ms_values**

   This array holds the calculated register values for the MSNx and MSx synthesizers.

4. **abcdef**

   This array holds the intermediate values that are used in the calculation. They provide visibility into the calculation.

5. **flag**

   Diagnostic enable flag. Set to non-zero to cause the routine to display some of its internal actions.

Listing 10: si5351a_calc-29

```
#define RFZERO_MSNx_P1      0
#define RFZERO_MSNx_P2      1
#define RFZERO_MSNx_P3      2
#define RFZERO_MSx_P1       3
#define RFZERO_MSx_P2       4
#define RFZERO_MSx_P3       5
#define RFZERO_RX_DIV       6
#define RFZERO_MSX_DIVBY4 7
```

These are the index names for the **abcdef** array.

The **RFZERO_MSNx_P**x variables are the register patterns for the Stage 1 MSNA and MSNB multisynth.

The **RFZERO_MSx_P**x variables are the register patterns for the Stage 2 MS0..MS2 multisynth.

The **RFZERO_RX_DIV** variable is the value for the divider on the output of the MS0..MS2 multisynth. It should always be zero, indicating divide-by-1.

The **RFZERO_MSX_DIVBY4** variable is the enable flag for the divide-by-4 on the output of the MS0..MS2 multisynth. It should always be zero, indicating the divide-by-4 function is not in use.

Listing 11: si5351a_calc-180

```
    //
    //  Start by calculating the output to input frequency ratio.
    //    We need this to come up with the VCO frequency
    //
    abcdef[VCO_MULT]    = fout / xtal;
```

First step is to determine the factor used by the first multisynth.
This is simply the desired output frequency divided by the SI5351 crystal frequency. This is the basis for the MSNA register values.

Listing 12: si5351a_calc-185

```
    //
    //  Get the VCO to run between 600MHz and 900MHz
    //                             ------     ------
    //    So get the product of the multiplier from above
    //    and an integer division of the VCO frequency
    //    (and a power-of-2, if possible)
    //
    abcdef[FB_DIV] = 0;
    abcdef[VCO_FREQ] = 0.0;
    //
    // Loop through even "abcdef[FB_DIV]" ratios
    //
    while(abcdef[VCO_FREQ]<=900.0E6){
        abcdef[VCO_FREQ] = xtal * abcdef[VCO_MULT] * abcdef[FB_DIV];
        if( (abcdef[VCO_FREQ]>=600.0E6) )
            break;
        abcdef[FB_DIV]+=2;
    }
```

Now we will come up with the VCO operating frequency.
We want it to be an even multiple of the output frequency so the MSA multisynth can operate in integer mode. As long as we can keep the VCO operating between 600MHz and 900MHz it will be happy.
As we're way down in the 2M band, this isn't a problem.

The MSZ multisynth also prefers to operate with an even value divisor, so we'll step through the even values to start.

Listing 13: si5351a_calc-204

```
    //  If we didn't find a "pretty spot", where the
    //    VCO will be happy, try again using odd divisors
    //
    if(abcdef[VCO_FREQ]>900.0E6){
        abcdef[FB_DIV] = 1;
        abcdef[VCO_FREQ] = 0.0;
        while(abcdef[VCO_FREQ]<=900.0E6){
            abcdef[VCO_FREQ] = xtal * abcdef[VCO_MULT] * abcdef[FB_DIV];
            if( (abcdef[VCO_FREQ]>=600.0E6) )
                break;
            abcdef[FB_DIV]+=2;
        }
    }
```

As the comment indicates, if we run this code, then we're going to end up with an ugly divisor (i.e. an odd number).

Since we're below 150MHz in the 2M band, this code should **never** run.

Listing 14: si5351a_calc-217

```
    //
    //  If the abcdef[VCO_FREQ] exceeds 900MHz, we've failed, and miserably at that
    //  we'll notice it later and zero out the control words...
    //
    if(flag){
        fprintf(stdout, "\n");
        fprintf(stdout, "        ");
        fprintf(stdout, "xtal %.3f  ", xtal/1.0E6);
        fprintf(stdout, "trgt %.3f  ", fout/1.0E6);
        fprintf(stdout, "divs %.3f  ", abcdef[FB_DIV]);
        fprintf(stdout, "mult %.3f  ", abcdef[VCO_MULT]);
        fprintf(stdout, "VCOf %.3f  ", abcdef[VCO_FREQ]/1.0E6);
        fprintf(stdout, "\n");
    }
```

Diagnostics. This emits a report if the flag variable is non-zero.

Listing 15: si5351a_calc-231

```
    //
    //  Here we're going to try to come up with the
    //  "Synthesis, Stage 1" register values
    //
    //  the "a" field is the integer portion of the
    //    xtal multiplier.  b/c is the fraction.
    //
    //    fa = modf(abcdef[VCO_MULT], &abcdef[AIDX]);
    //
    //   abcdef[AIDX].fa is the xtal multiplication factor
    //
    fa = modf(abcdef[VCO_MULT]*abcdef[FB_DIV], &abcdef[AIDX]);
```

the *modf* library routine is used to split the MSNA multisynth crystal clock multiplier into integer and fractional portions.

The **fa** variable will hold the fraction. It should be less than one or the library routine messed up.

The **abcdef [AIDX]** variable ends up with the integer portion.

Listing 16: si5351a_calc-243

```
    //
    //  fa is the fractional part, so calculate
    //  b/c = fa
    //
    switch(2){
        case 1:
            abcdef[BIDX] = fa * K1048575;
            abcdef[CIDX] = K1048575;
            break;
        case 2:
            abcdef[BIDX] = fa * Kvalue;
            abcdef[CIDX] = Kvalue;
            break;
    }
```

Refer to the Skyworks application note AN619 section 3.2 for details of the calculation used to produce the multisynth register values.

We now need the fractional portion of the divisor to be expressed as a numerator and denominator. The numerator in the [BIDX] variable and the denominator. in the [CIDX] variable. We have some flexibility here, so we can simply maximize the [CIDX] variable (case '1') or use something else, like the largest power of 10 that fits (case '2').

Using the largest power of 10 approach, the [BIDX] variable ends up being expressed as the fractional part shifted left (a decimal shift). Now the [BIDX] variable looks like the fractional part from above.

Listing 17: si5351a_calc-257

```
    //
    //  OK, now we have the "a", "b", and "c" numbers to
    //  calculate the register values.  AN619-3 S-3.2
    //
    abcdef[BCFRAC] = abcdef[BIDX]/abcdef[CIDX];
    abc_fvco = xtal * (abcdef[AIDX] + abcdef[BCFRAC]);
```

The fraction we built with [BIDX]/[CIDX] above is stuffed into **abcdef[BCFRAC]** so it is visible to the caller.

The **abc_fvco** variable is used in several parts of the Skyworks calculation, so we perform the calculation once so it doesn't get inadvertently altered by a crooked finger as the code file is edited.

The **abc_fvco** variable should have the same value as the **abcdef[VCO_MULT]** variable from earlier.

Listing 18: si5351a_calc-263

```
    //
    //  We will use this "floor" function twice, so do it once
    //    here so it doesn't get mangled accidentally later...
    //
    bc128_floor = floor(128.0 * abcdef[BCFRAC]);
```

We do this calculation once here and use it several times.

The *floor* routine simply returns the largest integral value that is not greater than the argument. This shifts the decimal point in the fraction to the right 7 bits.

Listing 19: si5351a_calc-268

```
    //
    //  Calculation for the 1st. Multisynth: AN619-3 S-3.2
    //
    MSNx_P[1] = 128.0 * abcdef[AIDX] + bc128_floor - 512;
    MSNx_P[2] = 128.0 * abcdef[BIDX] - (abcdef[CIDX] * bc128_floor);
    MSNx_P[3] = abcdef[CIDX];
```

0

Now we can build the values that will be loaded into the MSNA and MSNB registers.

Listing 20: si5351a_calc-274

```
    //
    //  OK< the 1st. Multisynth values are saved...
    //
    if(flag){
        fprintf(stdout, "    MUL   ");
        fprintf(stdout, "abcdef[AIDX](%.3f) abcdef[BIDX](%.3f) abcdef[CIDX
           ↪ ](%.3f)   ", abcdef[AIDX]+fa, abcdef[BIDX], abcdef[CIDX]);
        fprintf(stdout, "floor(%.3f)   ", bc128_floor);
        fprintf(stdout, "fvco(%.3f)   ", abc_fvco/1.0E6);
        fprintf(stdout, "\n");
        fprintf(stdout, "          ");
        fprintf(stdout, "MSNx_P[1] %.3f   ", MSNx_P[1]);
        fprintf(stdout, "MSNx_P[2] %.3f   ", MSNx_P[2]);
        fprintf(stdout, "MSNx_P[3] %.3f   ", MSNx_P[3]);
        fprintf(stdout, "\n");
    }
```

Listing 21: si5351a_calc-289

```
    //
    //  2nd. Multisynth calculations.
    //  Another instantiation of this block,
    //    so the calculations are the same, just
    //    using a different divisor.
    //  RENAME a, b, c to d, e, f so we don't
    //    overwrite previous work...
    //
    //  We've built upon the assumption that this
    //    Multisynth is operating in integer mode,
    //    so the d and e numbers specify zero.
    //
    abcdef[DIDX] = abcdef[FB_DIV];
    abcdef[EIDX] = 0;
    abcdef[FIDX] = K1048575;
    abcdef[EFFRAC] = abcdef[EIDX] / abcdef[FIDX];
```

The MS0, MS1, and MS2 multisynths are setup the same. The values they are loaded with are calculate din the same manner as the MSNA and MSNB multisynths above, but we know that the divisor is integer, so the [EIDX]/[FIDX] fraction will be zero.

The [DIDX], as should be evident, is simply the VCO divisor calculated earlier.

Listing 22: si5351a_calc-305

```
    //
    //  That floor function, from above...
    //
    def_floor = floor(128. * abcdef[EFFRAC]);
```

As we did above, the FLOOR calculation is done once here and used in several locations.

Listing 23: si5351a_calc-309

```
    //
    //  Calculation for the 2nd. Multisynth: AN619-5 S-4.1.2
    //    We're operating in the 2M band, so we WILL stay below 150MHz
    //       so make use of the same equation set using different data.
    //  Since this is an integer division, the [2] field should calculate
    //  as ZERO.  Later we will change the [3] field to 1 so it looks nice.
    //
    MSx_P[1] = 128.0 * abcdef[DIDX] + def_floor - 512;
    MSx_P[2] = 128.0 * abcdef[EIDX] - abcdef[FIDX] * def_floor;
    MSx_P[3] = abcdef[FIDX];
```

Now the numbers for the three multisynth registers can be calculated. This is the same as the MSNA/MSNB multisynths.

Note that the **MSx_P[1]** calculation shifts the bits up in the P1 register. For the 2M band, we will arrive at a divisor of 6.000 which all ends up in **MSx_P[1]** as a value of 0x100.
We end up with: *(128 * 6) - 512*.

Listing 24: si5351a_calc-319

```
    //
    //
    //
    if(flag){
        fprintf(stdout, "    DIV   ");
        fprintf(stdout, "d(%.3f) e(%.3f) f(%.3f)   ", abcdef[DIDX], abcdef[EIDX], abcdef[FIDX]);
        fprintf(stdout, "floor(%.3f)   ", def_floor);
        fprintf(stdout, "\n");
        fprintf(stdout, "          ");
        fprintf(stdout, "MSx_P[1] %.3f   ", MSx_P[1]);
        fprintf(stdout, "MSx_P[2] %.3f   ", MSx_P[2]);
        fprintf(stdout, "MSx_P[3] %.3f   ", MSx_P[3]);
        fprintf(stdout, "\n");
    }
```

Diagnostics.

Listing 25: si5351a_calc-333

```
    //
    //  Last minute sanity checks.
    //    If we try to drive the internal VCO past 900MHz
    //      we are operating out-of-spec, so indicate a
    //      problem by zero-ing out all the counters
    //
    //    If we are "in-spec" copy the results to
    //      the callers buyffers.
    //
    if(     (abcdef[VCO_FREQ]<=900.0E6) &&
            (abcdef[VCO_FREQ]>=600.0E6) ){
        //
        //  1st. Multisynth
        //
        ms_values[RFZERO_MSNx_P1]  = MSNx_P[1];
        ms_values[RFZERO_MSNx_P2]  = MSNx_P[2];
        ms_values[RFZERO_MSNx_P3]  = MSNx_P[3];
        //
        //  2nd. Multisynth
        //
        ms_values[RFZERO_MSx_P1]  = MSx_P[1];
        ms_values[RFZERO_MSx_P2]  = MSx_P[2];
        ms_values[RFZERO_MSx_P3]  = MSx_P[3];
```

Last sanity checks before we pass the calculated values back to the caller.

The sanity check is to see that the VCO frequency is within the range allowed by the SI5351.

Listing 26: si5351a_calc-356

```
        //
        //  2nd. Multisynth fraction control:
        //     we're integer divisionj, so fraction is
        //     set to ZERO by making [2]=0 and [3]=1
        //     i.e. 0/1 for a fractional part of the divisor
        //
        if(!ms_values[RFZERO_MSx_P2])
            ms_values[RFZERO_MSx_P3] = 1;
```

Now for a bit of cleanup. The MS0..MS0 multisynths are to be operated in integer mode, so the P2 and P3 registers need to loaded with zero. If the P2 register is zero, the the value of P3 may be any value (as its not used).
So if the calculated P2 value is zero (and it should be) the force the P3 value to 1 to make the displayed number smaller (i.e. only one character, rather than 5).

Listing 27: si5351a_calc-364

```
    //
    //  The R Dividers (AN619-6 S-4.2.2)
    //    are set to "divide by 1" using an
    //    index of 0.
    //  This is because we are operating
    //    well above the 500KHz point where these
    //    dividers become necessary.
    //
    ms_values[RFZERO_RX_DIV]   = 0;
```

Returning more diagnostic information.

Listing 28: si5351a_calc-373

```
    //
    //  The MSx_DIVBY4[1:0] is set to ZERO
    //    because we are operating below 150MHz
    //    (144MHz to 148MHz)
    //
    ms_values[RFZERO_MSX_DIVBY4]   = 0;
```

We are operating in the 2M band, well above the point where the divider in the clock output block would be needed, so it gets forced to zero.

Listing 29: si5351a_calc-379

```
    //
    //  An finally, pass the integer feedback
    //    divisor we discovered at the begining.
    //
    sts = abcdef[FB_DIV];
    //
```

Diagnostics.

Listing 30: si5351a_calc-385

```
    }else{                                                  385
        ms_values[RFZERO_MSNx_P1]      = 0;                 386
        ms_values[RFZERO_MSNx_P2]      = 0;                 387
        ms_values[RFZERO_MSNx_P3]      = 0;                 388
        ms_values[RFZERO_MSx_P1]       = 0;                 389
        ms_values[RFZERO_MSx_P2]       = 0;                 390
        ms_values[RFZERO_MSx_P3]       = 0;                 391
        ms_values[RFZERO_RX_DIV]       = 0;                 392
        ms_values[RFZERO_MSX_DIVBY4]   = 0;                 393
        sts = 0;                                            394
    }                                                       395
```

The beginning of the block is one line 342. If the target VCO frequency we calculated is out of range, pass all zeros back.

Listing 31: si5351a_calc-396

```
    return sts;                                             396
}   //  kc0jfq_setup()                                      397
```

Return the divisor we planned to stick into the MS0..MS0 multisynth.

13.2 ICS307

Insufficient stock on hand to fabricate the 102-73181-0 boards. No additional documentation at this time.

par

13.3 ICS525

Small stock of the ICS525 on hand to fabricate a few additional 102-73161-25 boards. The zNEO in the 80-pin package, however, is near impossible to find.

Existing 102-73161-25 boards will be supported with this software release.

14 Assorted Interesting Topics

This section deals with topics that don't fit nicely into the above sections.

Somewhat of a F.A.Q. if you will...

14.1 Transmitter Configuration

The transmitter may be built in of several configurations. The base board has a MAX2602 NPN transistor to provide a bit of gain. The ICS525 alone provides about 20mW of output. There are also a pair of patch boards that can be installed in place of the MAX2602 to provision with a MMIC type amplifier or a simple plastic package NPN transistor in SOT23.

14.1.1 Transmitter Configuration, Low Power

R31, C69, Q2, L1, L4, L5 and L7 are unpopulated. A haywire is then installed from the input side of L4 (near U2) to the output side of L7 (near L33). This bypasses the RF amplifier and its matching altogether.

This is the lowest power mode to configure the transmitter in.

14.1.2 Transmitter Configuration, MAX2602

Applies only to -12 artwork;

R31 is still unpopulated , C62, C67 and R12 are installed.

This is the configuration as originally designed.

14.1.3 Transmitter Configuration, MMIC

Applies to -12 artwork;

C21, C57, L4, L5, and Q2 are unpopulated. The MMIC patch board is installed on to the main board. L4 and L1 are then installed to connect the patch board to the main board. C21 and the MMIC can then be installed.

14.1.4 Transmitter Configuration, MMIC

Applies to -21 and -23 daughter boards for the -25 artwork;

These are daughter boards for the -25 ma inboard. These have input and output matching networks which should not be required.

These board also have attenuators at the input to the MMIC to set the input level as needed. They are in the from of a PI network to maintain 50Ω impedance.

14.1.5 Transmitter Configuration, NPN in SOT23 package

Applies only to -12 artwork;

C21, C57, L4, L5, and Q2 are unpopulated. The MMIC patch board is installed on to the main board. L4 and L1 are then installed to connect the patch board to the main board. C21 and the transistor can now be installed.

14.2 Frequency Selection -12 artwork

Frequency Selection is dependent on the specific VCMO selected for use with the unit and the control voltage applied to the VCMO to modulate the carrier.

The preferred VCMO is the SiTime SIT3701AC-13-33C-24.xxxxx which has a tuning range of +/- 60PPM. Typically the +/-120PPM part is found on DigiKey which requires changing R44 or R47 to a larger value.

14.3 Frequency Selection -25 artwork

Crystals added to the -25 artwork to eliminate the need to deal with the surface mount oscillators. The clock generator crystal has varicaps to push the crystal frequency. The same 20MHz crystal may be used for both the NEO and the ICS525.

To allow the variable oscillator to settle close to it nominal frequency, select crystals that expect to see a higher load capacitance.

14.4 Lost Sequences

You may find that some sequences seem to *get lost* when running multiple sequences. This will occur when the timing characteristics are not correctly taken into account.

The processor in the Fox Transmitter is not running a real-timer O/S, rather it is a simple loop that checks for work that needs to be done every 100 milliseconds. When the starting point for a schedule occurs, that particular schedule is run to completion. Any other schedules that should have run when this schedule is running will be ignored.

The solution, of course, is to carefully time these schedules to execute serially. Parallel execution is not supported (and that notion doesn't make sense).

Correctly timing the master schedule takes advantage of the behavior to suppress time update messages during message traffic. To achieve this, simply have the **MAS** schedule offset set to 5 and the other schedules start at 0.

14.5 Notes on the use of the Network Port

The network port is provided as a quick means of locking multiple units together in the field. Time synchronization is also achieved using the flash memory loading utility.

A group of units may have their time set using the flash memory loading utility the night before a fox hunt to eliminate the need to have interconnect cables the day of the hunt. The TOY clock will keep track time to within a few seconds per day which should be close enough for fox hunt operations.

14.6 Prosigns

How do you go about creating a prosign?.

A prosign is a group of letters that are run together. These are typically shown with an overbar in various documents. They consist of several letters strung together wit abnormal inter-character timing.

You have control over the timing characteristics using the **CWPM** command. Nominally the timing weights are 1, 3, 5, and 7 as discussed in the section above. Altering these timing weights to 1, 1, 5, 7 will, in effect, turn subsequent letter groups into prosigns.

This will probably require breaking the message up and interspersing the **CWPM** commands to switch timings. The effect of the **CWPM** command is immediate, making control as described possible.

14.7 Code Speed of the ID message

Some of the examples have messages at wildly varying word rates. This may make it difficult to identify individual stations. To address this, the examples send the ID message at the beginning and end of a transmission at 20 words-per-minute. The speed of the message content may then change to any desired rate.

One of the examples varies the word rate throughout the message, starting out a 35 WPM and ending at 15 WPM. Variations on this are straightforward to sequence (see the CWPM command on page 117).

14.8 External Transmitter Control

The external transmitter control feature shares the audio tone generator with the internal transmitter. Should you choose to key the internal transmitter at the same time as the external transmitter, you are stuck with the same tone on both. Driving them at different time, however, you are free to alter the audio pitch as desired.

14.9 External Transmitter Serial Control

No control software is present to deal with controlling an external transmitter. There is, however, diminishing room in the zNEO program flash to implement such control.

The network port can be used for such control if the radio can deal with logic level (i.e. 3.3 Volt) signals. As built the network port presents levels seen at the UART pins on the zNEO package. This polarity is opposite of that send on an RS232 interface. In other words, the line idles high, at 3.3V. A start bit is represented as 0V.
The polarity if controlled by U5, which is normally populated with a 74LVC2G07 which is an open drain bob-inverting buffer. This device could be switched for a 74LVC2G14 which is the inverting gate **without** the open drain. The transmit channel and receive channel must, then, be kept separate. R13 and R14 may need to be replaced with small value capacitors to shunt any RF pickup (so keep the interconnect short and shielded).

14.10 Controlling Deviation

The following applies directly to the -25 revision boards.

Deviation Control uses two pins on the zNEO, one is an enable and the other is the output of Timer 0. The timer should be programmed to generate a square wave at the desired audio frequency and left running continuously. zNEO pin PA0 is the used to gate the clock through a tri state buffer. When the buffer is tri-state, the termination network R10/R11 should be configured to set the clock at its nominal frequency. Gain is set by R47 and R10/R11. The gain should be selected to limit the deviation to what the receiving station can handle, typically limiting the RF output deviation to 2KHz.

R47/R44 and C51 form the audio low pass filter. It removes the high frequency content from the square wave supplied by the zNEO.

The tri-state buffer, U10, attempts to center the carrier on the nominal transmit frequency. When generating a tone the carrier is modulated about the center frequency. Lacking the tri-state buffer, the modulation would occur only above or below the carrier.

14.11 xxx

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

14.12 xxx

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

15 Actual FOX configuration commands

This is a listing of typical fox commands.
The REM-012234567890123456789012234567890 lines show limits of the 32 byte FRAM record.
Lines without the *esav* are not placed in FRAM.

Updates in the fox operating software and the support utilities allow command argument substitution. Should feel familiar to a bash shell user.

15.1 102-73161-25 setup scripts

These scripts are taken from working units.

15.1.1 FOX8_W0JV.fox

Setup device identity.
Multiple schedules are loaded, but left inactive.

Listing 32: FOX8_W0JV.fox

```
REM– 01234567890123456789012234567890
esav REM– ./fox_simple –F FOX8_W0JV
#
#       Our Epoch is CDT: –5 Hours from Zulu
#       Set system time from DS1672
#
esav INI=EPOC –5.0
esav INI=TSET
#
#       Setup
#             Callsign & Nickname
#
esav INI=NAME FOX8
esav INI=CALL W0JV/8
#
# Call Common initialization
#             $1 INI        File Name
#             $2 144.150    Announce Frequency
#             $3 REV_25     Hardware Revision
#             $4 20.0       RF Xtal Frequency
#             $5 450        Schedule offset
#
#include fox_init_300.fox INI 144.150 REV_25 20.0 255 25
#include /home/wtr/Fox_Transmitter/trunk/FOX8_W0JV_AUDIO.fox
#
#       Operating Schedules
#
#include fox_announce.fox ANN 144.285
#include text.fox S0
#include emmett.fox S2
```

```
#include fox_5.fox S5
#include batt_2.fox S8
#include batt_3.fox S9
```

15.1.2 fox_init_300.fox

Common Initialization for a 300 second cycle.

Listing 33: fox_init_300.fox

```
#
#        Common initialization
#              $1 INI         File Name
#              $2 144.150     Announce Frequency
#              $3 REV_25      Hardware Revision
#              $4 20.0        RF Xtal Frequency
#              $5 450         Schedule offset
#              $6 450         Ghost "S2" offset
#
#------------------------------------------------
#
REM-- ########################
REM-- ## 5 min cyc, 4 units  ##
REM-- ## .1 $1      Name      ##
REM-- ## .2 $2 Ann Frq  ##
REM-- ## .3 $3  Hard Rev ##
REM-- ## .4 $4     RF Xtal  ##
REM-- ## .5 $5      Sch off  ##
REM-- ## .6 $6      "S2" off ##
REM-- ########################
#
#------------------------------------------------
#
#        CONF commands MUST BE FIRST!!!
#        some followup commands will NOT decode correctly
#          prior to defining the hardware configuration
#
esav $1=CONF $3
esav $1=CONF BUZZ
esav $1=CONF TONE
#
#------------------------------------------------
#
#  Once the hardware environment is defined,
#       the VCO/PLL reference can be defined
#       and we can program the ICS525/ICS307/DRA818/SA818
#
esav $1=XTAL $4
esav $1=TOYC 2K
esav $1=FREQ $2
#
```

```
#       Set schedules,  leaving ONLY
#         the "S2" and "S5" active.
#         others appear in flash so they can be
#         easily activated using "EZER <nn>" and
#         "ESAV MODS <Sx> <per> <off>
#
REM– 012345678901234567890123456789 0
esav $1=REM– STRT 00:00:00
esav $1=REM– MODS S0 20:00 01:00
esav $1=REM– MODS S1 10:00 03:30
esav $1=MODS S2 300 $6
esav $1=REM– MODS S3 10:00 09:30
esav $1=REM– MODS S4 10:00 08:30
esav $1=MODS S5 300 $5
esav $1=REM– MODS S9 10:00 00:30
```

15.1.3 fox_announce.fox

Send the startup announcement to the operator.

If there is an audio directory present and it contains files for the callsign and unit name, they will be verbalized when the unit is powered on so the hunt operator knows the unit is alive.

The *TALK CALL* command substitutes the callsign defined by the **CALL** command. Similarly, the *TALK NAME* substitutes the units nickname defined in the **NAME** command. This allows this announce sequence to be used in all the fox setups.

The *$1=* performs a similar substitution, replacing the *$1* with the 1$^{st.}$ argument from the calling sequence. The The *$1=FREQ $2* replaces the *$2* with the 2$^{nd.}$ argument (as well as replacing *$1*)..

Listing 34: fox_announce.fox

```
#
#       Upgrade to V1.50 software !!!
#       Annonce We're ready
#       Then reset to our operating frequency
# Startup Announce Message
#             $1 ANN        File Name
#             $2 144.150    Operating Frequency
#
REM– ########################
REM– ## Setup Announce Mssg ##
REM– ## .1 $1      Name      ##
REM– ## .2 $2 Op Frq    ##
REM– ########################
#
esav $1=TONE 1.0
esav $1=CWPM 20,-1,-1,-1,-1
esav $1=BEGN
esav $1=TALK <CALL>
esav $1=TALK <NAME>
esav $1=BATT 7.2
esav $1=BATS 7.2
esav $1=DONE
#
# Set Operating Parameters
#
esav $1=FREQ $2
esav $1=TONE 1.6
esav $1=CWPM 15
esav $1=STAT
```

15.1.4 FOX8_W0JV_AUDIO.fox Directory

These are the directory records for two very brief audio clips that announce a callsign and unit ID.

Note the last column in the last line is the lowest address in FRAM used by the waveform data. The first numeric column on that line is the starting record number.

Listing 35: FOX8_W0JV_AUDIO.fox

```
#
#         V0.3
#         MAKE 2.10
#         2021-02-19T17:32:12
#    Invocation: ./pwm_audio_util -FFM25V02A FOX8_W0JV_AUDIO.txt
#       Control: FOX8_W0JV_AUDIO.txt
#     Directory: FOX8_W0JV_AUDIO.fox
#      Download: FOX8_W0JV_AUDIO.hex
#        Device: FM25V02A    Cypress
#          Size: 256Kb
#       Records: 1024
#
# Audio File Directory Structure
#
#    esav TALK=<name> <rec num> <rec cnt> <rec addr>
#
#   <name>      TALK file name ("TALK <name>" to play)
#   <rec num>   flash file system record number
#   <rec cnt>   number of records in audio clip
#   <rec addr>  flash file system record Address
#                    this matches up with the HEX file
#
#    0123456789012345678901234567890
esav REM- Audio File Directory
REM-
REM-    Each audio record is 32 samples
REM- and is contained in a single HEX
REM- record.
```

```
REM-   Type-4 extended address records occur
REM- regularly throughout the file.  More
REM- often than is actually required.
REM-
REM- An Intel HEX record consis of
REM-     :       delimiter
REM-     20      length, 8 bits in 2 char
REM-     0000    address 16 bits in 4 char
REM-     00      type, 8 bits in 2 char
REM-     xx..xx data, 'length' 2 char values
REM-     00      checksum, 8 bits in 2 char
REM-   Record Types
REM-     00      Data
REM-     01      EOF
REM-     04      extended address
REM-               data field has A31..A16
REM-
esav TALK=W0JV 833 191 0x6820
esav TALK=FOX8 745 88 0x5D20
esav TALK=EMMETT 690 55 0x5640
```

15.1.5 text.fox

Test the audio bandwidth of the transmitter.
Argument 1 from caller is file name, 0 through 9.

The *$1=* substitution operation replaces the *$1* with the 1st. argument from the calling sequence. The argument in this example is "S0".

Listing 36: text.fox

```
#
#        Audio Bandwidth Test
#
REM- ########################
REM- ## Audio Bandwidth Tst ##
REM- ## .1 $1      Sch Name  ##
REM- ########################
REM- 012345678901223456789 0123
#
#
#  Signon message.
#
esav $1=REM- text.fox
esav $1=TONE 1.0
esav $1=CWPM 25,-1,-1,-1,-1
esav $1=BEGN
#
esav $1=TONE 0.2
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 0.3
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 0.4
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 0.5
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 0.6
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 0.7
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 0.8
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 0.9
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 1.0
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 1.1
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 1.2
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 1.3
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 1.4
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
```

```
#
esav $1=TONE 1.5
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 1.6
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 1.7
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 1.8
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 1.9
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 2.0
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 2.1
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 2.2
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 2.3
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 2.4
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
#
esav $1=TONE 2.5
esav $1=CODE hi hi hi hi hi hi
esav $1=WAIT 1
# Signoff
#
esav $1=CWPM 25,-1,-1,-1,-1
esav $1=TONE 1.0
esav $1=DONE
```

15.1.6 emmett.fox

Isolated text utterance over-the-air.

This simply keys the transmitter and allows the RF to stabilize and then sends a short audio clip the stops transmitting.

Listing 37: emmett.fox

```
REM– #########################
REM– ## Say Emmetts Name      ##
REM– ## .1 $1       Sch Name  ##
REM– #########################
REM– 0123456789012234567890123
#
esav $1=REM– emmett.fox 1–Sec
#
#  Signon message.
#
esav $1=RFON
esav $1=WAIT 1
esav $1=TALK emmett
esav $1=WAIT 1
esav $1=RFOF
```

15.1.7 fox_5.fox

The *$1=* substitution operation replaces the *$1* with the 1$^{st.}$ argument from the calling sequence. The argument in this example is "S5".

Listing 38: fox_5.fox

```
REM– #########################
REM– ## Schedule              ##
REM– ## .1 $1       Sch Name  ##
REM– #########################
REM– 0123456789012234567890123
#
#
#   ICARC operating sequence
#
esav $1=REM– fox_5.fox 155–Sec
#
#  Signon message.
#
esav $1=CWPM 20,–1,–1,–1,–1
esav $1=BEGN
esav $1=TALK <call>
esav $1=TALK <name>
esav $1=CWPM 20,–1,–1,–1,–1
esav $1=TONE 0.880
esav $1=BATT 7.2
esav $1=BATS 7.2
esav $1=TONE 1.4
#
#   Fill time so they have a chance of finding me
#
esav $1=CWPM 20,-1,-1,-1,-1
esav $1=WAIT 3
esav $1=CODE IOWA CITY
esav $1=CODE AMATEUR RADIO
esav $1=CODE CLUB FOX HUNT
esav $1=WAIT 1
esav $1=CODE This is a test
esav $1=CODE of the code
esav $1=CODE generator in the
esav $1=CODE KC0JFQ FOX TX
esav $1=CODE built for the
esav $1=CODE IOWA CITY
esav $1=CODE AMATEUR RADIO
esav $1=CODE CLUB FOX HUNT
esav $1=WAIT 3
esav $1=CWPM 20,-1,-1,-1,-1
#
# Signoff
#
esav $1=DONE
```

15.1.8 batt_2.fox

The *$1=* substitution operation replaces the *$1* with the 1$^{st.}$ argument from the calling sequence. The argument in this example is "S8".

Listing 39: batt_2.fox

```
#
#       Test the battery monitor and audio bandwidth
#
REM- #########################
REM- ## Battery Monitor     ##
REM- ## .1 $1      Sch Name ##
REM- #########################
REM- 01234567890122345678901 23
#
#
#  Signon message.
#
esav $1=REM- $1_text.fox
esav $1=TONE 1.0
esav $1=CWPM 25,-1,-1,-1,-1
esav $1=BEGN
#
esav $1=CWPM 24,-1,-1,-1,-1
esav $1=TONE 0.2
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 12,-1,-1,-1,-1
esav $1=TONE 0.3
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 13,-1,-1,-1,-1
esav $1=TONE 0.4
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 28,-1,-1,-1,-1
esav $1=TONE 0.5
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 22,-1,-1,-1,-1
esav $1=TONE 0.6
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 26,-1,-1,-1,-1
esav $1=TONE 0.7
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 19,-1,-1,-1,-1
esav $1=TONE 0.8
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 21,-1,-1,-1,-1
esav $1=TONE 0.9
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 15,-1,-1,-1,-1
esav $1=TONE 1.0
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 18,-1,-1,-1,-1
esav $1=TONE 1.1
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 30,-1,-1,-1,-1
esav $1=TONE 1.2
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 23,-1,-1,-1,-1
esav $1=TONE 1.3
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 24,-1,-1,-1,-1
esav $1=TONE 1.4
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 20,-1,-1,-1,-1
esav $1=TONE 1.5
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 14,-1,-1,-1,-1
esav $1=TONE 1.6
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 11,-1,-1,-1,-1
esav $1=TONE 1.7
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 29,-1,-1,-1,-1
esav $1=TONE 1.8
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 16,-1,-1,-1,-1
esav $1=TONE 1.9
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 10,-1,-1,-1,-1
esav $1=TONE 2.0
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 17,-1,-1,-1,-1
esav $1=TONE 2.1
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 28,-1,-1,-1,-1
esav $1=TONE 2.2
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 27,-1,-1,-1,-1
esav $1=TONE 2.3
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 24,-1,-1,-1,-1
esav $1=TONE 2.4
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=CWPM 16,-1,-1,-1,-1
esav $1=TONE 2.5
esav $1=BATT 7.2
esav $1=WAIT 1
#
# Signoff
#
esav $1=CWPM 25,-1,-1,-1,-1
esav $1=TONE 1.0
esav $1=DONE
```

15.1.9 batt_3.fox

The *$1=* substitution operation replaces the *$1* with the 1st. argument from the calling sequence. The argument in this example is "S9".

Listing 40: batt_3.fox

```
#                         Test of time message.
#                         NOT SCHEDULED!!!
#
REM- ########################
REM- ## Battery Monitor    ##
REM- ## .1 $1     Sch Name ##
REM- ########################
REM- 0123456789012234567890123
#
#
#  Signon message.
#
esav $1=REM- $1_text.fox
esav $1=TONE 1.0
esav $1=CWPM 25,-1,-1,-1,-1
esav $1=BEGN
#
esav $1=TONE 0.2
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 0.3
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 0.4
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 0.5
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 0.6
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 0.7
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 0.8
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 0.9
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.0
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.1
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.2
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.3
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.4
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.5
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.6
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.7
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.8
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 1.9
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 2.0
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 2.1
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 2.2
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 2.3
esav $1=BATS 7.2
esav $1=WAIT 1
#
esav $1=TONE 2.4
esav $1=BATT 7.2
esav $1=WAIT 1
#
esav $1=TONE 2.5
esav $1=BATS 7.2
esav $1=WAIT 1
# Signoff
```

```
#
esav $1=CWPM 25,−1,−1,−1,−1
esav $1=TONE 1.0
esav $1=DONE
```

15.2 102-73181-10 setup scripts

This is the current master setup for the FOX20 through FOX25 transmitters. The **fox_simple** utility has some additions to make sharing these *fox scripts* a bit easier.

Every effort has been made to keep the commanding identical to the previous version to avoid causing old scripts to become obsolete.

15.2.1 FOX2X_KC0JFQ.fox

Listing 41: FOX2X_KC0JFQ_1

```
1  REM– 67890123456789012234567891
2  esav REM– ./fox_simple
3  esav REM– −S TACH
4  esav REM– −F FOX2X_KC0JFQ.fox
5  #
6  #  /home/wtr/Radio/halo_term/fox_simple –STACH −c250 −t10 –fFOX20_KC0JFQ.fox
7  #  /home/wtr/Radio/halo_term/fox_simple –STACH −c100 −t10 −f/home/wtr/WAV/
      ↪ fox_73181_r4k.hex
8
9  #  Limited voice storage
10 #
11 #include talk_73181.fox
12 #include freq_5351.fox
```

Header lines, that document what we're up to.

The include directives (lines 11 and 12) tells *fox_simple* to insert the talk directory and supplemental frequency table at this point. This is a convenience for us in managing files. A command file can be inserted anywhere it is needed.

Listing 42: talk_73181.fox

```
esav TALK=BATTI 0
esav TALK=BATTV 4224
esav TALK=REG5 8704
esav TALK=POINT 13824
esav TALK=V_HZ 15232
esav TALK=V_KHZ 17664
esav TALK=V_MHZ 20864
esav TALK=V_N0 24064
esav TALK=V_N1 26752
esav TALK=V_N2 28544
esav TALK=V_N3 30720
esav TALK=V_N4 32640
esav TALK=V_N5 34560
esav TALK=V_N6 36736
esav TALK=V_N7 38528
esav TALK=V_N8 40448
esav TALK=V_N9 41984
esav TALK=V_MAMP 44416
esav TALK=V_VOLTS 48128
esav TALK=KC0JFQ 51200
esav TALK=W0JV 56960
esav TALK=FOX20 63104
esav TALK=FOX21 66048
esav TALK=FOX22 70272
esav TALK=FOX23 74624
esav TALK=FOX24 79872
esav TALK=FOX25 84608
esav TALK=FOX26 90752
esav TALK=FOX27 95744
esav TALK=FOX28 100352
esav TALK=FOX29 104832
esav TALK=V_F144 109824
esav TALK=V_F145 114560
esav TALK=V_F200 120576
esav TALK=V_F225 126336
esav TALK=V_F250 132352
esav TALK=V_F275 138752
esav TALK=V_F300 144640
esav TALK=V_FOX30 150528
esav TALK=V_FOX31 154368
esav TALK=V_FOX32 159104
esav TALK=V_F325 164352
esav TALK=V_F350 169088
esav TALK=V_F375 173696
```

This is a list of the audio clips that are loaded into FLASH memory. This assumes that software version **V3.27** of later is loaded into the fox transmitter. Sample rate and sample count are extracted from the RIFF/WAVE header located at the specified address.

Listing 43: freq_5351.fox

```
REM-  Call Line: "/home/wtr/Fox_Tx_73181/trunk/si5351a_calc -T2 -F
   ↪ 144.250,144.300,5 -o-14 -e freq_5351.fox "
REM- Output File: freq_5351.fox
REM- SI5351 Xtal:  20.000MHz
REM- Freq Offset: -14.000KHz
esav 144.255=13A2,D0980,F4240
esav 144.260=13A3,0B540,F4240
esav 144.265=13A3,3A340,F4240
esav 144.270=13A3,6913F,F4240
esav 144.280=13A3,C6D40,F4240
esav 144.285=13A4,01900,F4240
esav 144.290=13A4,30700,F4240
esav 144.295=13A4,5F500,F4240
esav=144.375=13A7,70E40,F4240
```

This is a supplemental frequency table. It is used in this example to add several frequencies not found in the internal table.

Listing 44: FOX2X_KC0JFQ_13

```
#
#       Our Epoch is CDT: -5 Hours from Zulu
#       Set system time from DS1672
#
esav INI=NAME 'name'
esav INI=CALL 'call'
esav INI=TIME
esav INI=EPOC -5.0
#
#esav INI=CONF BMON 12.5V
esav INI=CONF SI5351
esav INI=CONF 8MA CLK0
esav INI=CONF DRA818
esav INI=FREQ 144.150
#
```

This illustrates parameter substitution that was added to the *fox_simple* utility. The **name** and **call**sign are changed to values provided to textitfox_simple utility on the command line.

This allows a single script to be used to configure all the fox transmitters in a group.

This **TEST=** sequence is run when the *TEST* jumper is in place. This basic TEST configuration should **never** command the fox to transmit. This should be simple recovery commands.

Listing 45: FOX2X_KC0JFQ_28

```
#       Set schedules,  leaving ONLY
#
REM- 0123456789012345678901234567890
esav INI=MODS S0 'run'
esav INI=MODS S1 30 0
#
esav TEST=CWPM 35,-1,-1,-1,-1
esav TEST=CONF
esav TEST=STAT
#
esav MAS=CWPM 35,-1,-1,-1,-1
esav MAS=STAT
#

#
esav ANN=REM- fox_ann_V2023.fox
```

This **INI=** sequence is run when the TEST and MAS jumpers are removed.

Now we see a normal initialization sequence; making use of the *<name>* and *<call>* substitutions provided by *fox_simple* to use this script to load all of the 102-63181-10 fox transmitters.

The parameter substitution mechanism is used again here to replace the **'name'** and **'call'** fields, as was done in the *TEST=* section.

The **TIME** command, lacking any arguments, simply copies the hardware clock into the system clock. *POOF*! our transmitter is now running on truncated UT.

The **EPOC** command sets the local timezone offset for Central Daylight time (5 hours before ZULU).

And on to the CONF commands. This setup is used in testing so we play some games to allow things to change without having to reload the FRAM. Both **CONF DRA818** and **CONF SI5351** are present to allow one to be removed (**eras <nn>**). The **CONF 8MA CLK2** is superfluous for the DRA818 and id simply ignored (no harm, no foul).

When we are using the SI5351, the **CONF 8MA CLK2** enables the CLK2 output and sets the drive strength to, in effect, full. This provides a clock to the 102-73161-29 low power amplifier daughter board. The RF clock is delivered through a LVDS driver on the motherboard to the corresponding LVDS receiver on the amplifier board. Selecting **8MA** provides a 50Ω output impedance from the SI5351.

The next step is to select the transmitter frequency. The selection here is the common setup frequency, **not** the operating frequency.

The fox transmitter will send an *aliveness* message when it is powered on to allow the setup operator to hear that things are, indeed, working during hunt setup.

We will switch to our operating frequency at some later time.

More-or-less the final step is to set the operating schedule. Each transmitter will run with the same cycle time, but with a different offset. The offsets begin spread evenly through the specified period.

If you have set things up correctly, the message transmission time fits within the allotted period.

The schedule, being unique for each unit, is set through the substitution feature of the *fox_simple* utility.

Listing 46: FOX2X_KC0JFQ_44

```
esav ANN=TONE 1.0                 44
esav ANN=CWPM 30,-1,-1,-1,-1      45
esav ANN=BEGN                     46
esav ANN=TALK <CALL>              47
esav ANN=TALK <NAME>              48
esav ANN=WAIT 1.0                 49
esav ANN=BATV V                   50
esav ANN=BATV I                   51
esav ANN=WAIT 0.3                 52
esav ANN=TALK 'freqM'             53
esav ANN=TALK 'freqK'             54
esav ANN=DONE                     55
esav ANN=FREQ 'freq'              56
esav ANN=STAT                     57
esav ANN=RUN0 S0                  58
#                                 59
#       Operating Schedules       60
```

This **ANN=** sequence is run after the **INI=** sequence. This is the station aliveness message.

We make use of the *'freq'* substitution here to set the operating frequency.

The basic setup for operation on lines 43 through 45. The audio CW tone is set on line 42 and the chipping rate on line 43. This setup would typically have a rather fast chipping rate to reduce on-air time as we drop transmitters off for the hunt.

The **BEGN** command enables the transmitter and send out a CQ call using the station callsign previously stored by the **CALL** command .

We then proceed to verbalize the callsign and nickname for the setup operator to verify the correct station is being dropped.

The **BATV** commands report the voltage and current for the transmitter, again so the operator can be confident the fox transmitter will remain functional throughout the hunt.

The **DONE** is used to finish up a message transmission. The station callsign is again sent in a short *SK* message to keep the transmitter in compliance with the rules.

After that message is sent the transmitter will go quiet. Power is removed from the daughter board.

The station is now quiet, and the frequency is changed to the operating frequency. Assuming no additional frequency setting commands occur, the transmitter parameters are set.

The last command in the announce message is to enable schedule zero. This is the only schedule loaded into the system in this example, so we issue the command to enable the schedule.

The transmitter now looks at the current time and the schedule table once per seconds waiting for the appropriate time to deliver the scheduled message.

Listing 47: FOX2X_KC0JFQ_61

```
#        The power draw of the PS may cause the
#        "BATC" form of the report to take too long, so we
#        ALWAYS use the vocal report.
#
esav S0=TONE 1.0
esav S0=CWPM 20,-1,-1,-1,-1
esav S0=BEGN
esav S0=TALK <call>
esav S0=TALK <name>
esav S0=WAIT 0.5
#esav S0=BATV V
#esav S0=BATC EV
#esav S0=WAIT 0.5
#
#    Fill time so they have a chance of finding me
#
esav S0=TONE 1.5
esav S0=CWPM 25,-1,-1,-1,-1
esav S0=WAIT 0.15
#esav S0=CODE hi hi hi
esav S0=BATC EV 7.2
esav S0=WAIT 0.5
esav S0=CODE IOWA CITY
esav S0=CODE AMATEUR RADIO
esav S0=CODE CLUB FOXHUNT
esav S0=CODE F W KENT PARK
#esav S0=CODE MARCH 30 2024
#
#
# Prepare (kinda...) for Signoff
#     these extr TONE commands can be deleted and replaced
#     with more CODE commands to adjust time...
esav S0=TONE 1.0
esav S0=TONE 1.0
esav S0=TONE 1.0
#
# Signoff
#
esav S0=TONE 1.0
```

Here lies the **S0** message that is sent out periodically.

Although the previous commands in the **ANN=** section set audio tone and chipping rate, these are set at the beginning of the **S0** message. In this example, a different chipping rate is specified using standard weightings. All this can be changed in the **CWPM** command.

The audio tone frequency is moved slightly from the **ANN=** section. Allowed, of course, but not required.

The same **BEGN** command enables the transmitter and send a **CQ** message with the station callsign.

This setup verbalizes the station identity, both because we can (as we have the capability) and to make life easy for the fox hunters (so they can stay aware of station identity).

We also verbalize battery status, just like we did in the **ANN=** section. Hunt organizers can casually listen in on this traffic to track the health of the transmitters.

Battery status could as easily been sent in code using the **BATC** command.

Notice the **WAIT** commands that just cause a bit of dear air (with carrier). Our example here give us about 500mS of silence.

At line 72, we start to be real bastards to our diligent hunters by changing the audio frequency and the chipping rate. Our station now sounds a bit different than it did when the message traffic began.

The **CODE** commands send the text fragments (in our example "IOWA CITY AMATEUR RADIO"). Text is broken up so that it fits within the 31 byte limit of each command record.

When finished, we use the **DONE** command to clean-up at the end. Here again, we mess with the audio tone and chipping rate to match the feel of the signon message at the beginning.

The **ONCE** command is used to time the message as it is sent out. We see the command reports as the sequence runs. The

```
once s5=

sts00,371* ONCE: One Time Sequence Test "S5="
sts20,00* Handler_WPMR (cmd_code.c*) Handler_CWPM 25 (1 3 7 14)  0.04 Sec
sts23,23* Handler_BEGN (cmd_message.c*) 18:40:59.120  "e.. CQ CQ CQ DE KC0JFQ "  BEGN    11.20 Sec
sts20,00* Handler_WPMR (cmd_code.c*) Handler_CWPM 20 (1 3 7 14)  0.04 Sec
sts28,17* Handler_BATC (cmd_battery.c*)  BATC HI HI 7.656  9.04 Sec
sts26,00* Handler_WAIT (cmd_message.c*) TIMER_delay_ticks(50);  0.51 Sec
sts29,00* Entry_AUDIO_ (cmd_voice.c*)  TALK=BATTV,4224,4416,4K
sts29,00* Entry_AUDIO_ (cmd_voice.c*)  TALK=V_N7,38272,1888,4K
sts29,00* Entry_AUDIO_ (cmd_voice.c*)  TALK=POINT,13824,1344,4K
sts29,00* Entry_AUDIO_ (cmd_voice.c*)  TALK=V_N7,38272,1888,4K
sts29,00* Entry_AUDIO_ (cmd_voice.c*)  TALK=V_VOLTS,47744,2944,4K
sts29,05* Handler_BATV (cmd_battery.c*)  7.66V  3.37 Sec
sts26,00* Handler_WAIT (cmd_message.c*) TIMER_delay_ticks(50);  0.51 Sec
sts29,00* Entry_AUDIO_ (cmd_voice.c*)  TALK=BATTI,0,4160,4K
sts29,00* Entry_AUDIO_ (cmd_voice.c*)  TALK=V_N4,32384,1888,4K
sts29,00* Entry_AUDIO_ (cmd_voice.c*)  TALK=V_N9,41728,2304,4K
sts29,00* Entry_AUDIO_ (cmd_voice.c*)  TALK=V_MAMP,44032,3616,4K
sts29,04* Handler_BATV (cmd_battery.c*)  49mA  3.20 Sec
sts26,00* Handler_WAIT (cmd_message.c*) TIMER_delay_ticks(50);  0.52 Sec
sts20,00* Handler_WPMR (cmd_code.c*) Handler_CWPM 25 (1 3 7 14)  0.04 Sec
sts28,28* Handler_BATC (cmd_battery.c*)  BATC SOS SOS TTTTTTT EEEEEE  8.73 Sec
sts28,26* Handler_BATC (cmd_battery.c*)  BATC HI HI TTTTTTT EEEEEE  7.39 Sec
sts27,19* Handler_DONE (cmd_message.c*) 18:41:43.770  "DE KC0JFQ SK SK SK "  9.00 Sec
sts00,13* Execution Time: 53.760
RDY00,00* (Sp=0xBFA0)+1932 18:41:52.820
```

Listing 48: fox20.fox_249

Here are some commands that are saved for later reuse. Because the sequence number is not 0..9, it can't be run by the scheduler.

16 Sample Output

Sample output.

16.1 Sample HELP

Example Help listing.

Listing 49: FOX_ICARC_help.txt

```
20:28:51: sts01,00* TEST HELP **  TEST HELP **  TEST HELP **
20:28:51: sts01,00* Idx MNE  Class     -------Arguments-------  Command Function
20:28:51: sts01,01*   1 HELP SYS                                Help Menu and Items
20:28:51: sts01,02*   2 HELP SYS       <string>                 matching help items
20:28:51: sts01,03*   3 ONCE SYS       <name>                   Test run the named seqwuence
20:28:51: sts01,04*   4 REM- SYS                                Remark, (side-effect: stops schedules)
20:28:51: sts01,05*   5 RUN0 SYS                                RUN ALL Schedules
20:28:51: sts01,06*   6 RUN0 SYS       <name>                   RUN Specific Schedule
20:28:51: sts01,07*   7 IDLE SYS                                STOP ALL Schedules
20:28:51: sts01,08*   8 STAT SYS       <flag>                   System Status, (I)dent scan
20:28:51: sts01,09*   9 CONF SYS       <keywords>               Hardware Configuration
20:28:52: sts01,10*  10 TOYC SYS       <res> (250 2K 4K NONE)   Hi chg rte DS1672 bat
20:28:52: sts01,11*  11 TIME SYS       <time value>             Set Time (set DS1672)
20:28:52: sts01,12*  12 TIME SETUP                              Time from DS1672 to System (NO Argument!)
20:28:52: sts01,13*  13 EPOC SETUP     <hours>                  Epoch offset (i.e. time zone)
20:28:52: sts01,14*  14 CALL SETUP     <call>                   FCC Assigned Callsign
20:28:52: sts01,15*  15 NAME SETUP     <nick>                   Local Nickname
20:28:52: sts01,16*  16 NICK SETUP     <nick>                   alias for "NAME", but don't use it!
20:28:52: sts01,17*  17 TONE PGM       <freq>)                  Audio Tone (in KHz)
20:28:52: sts01,18*  18 CWPM PGM       <wpm gap1 gap2 gap3>     CW Chipping Rate
20:28:52: sts01,19*  19 FREQ PGM       <freq>                   Frequency (in MHz)
20:28:52: sts01,20*  20 5351 PGM       <key>,<value>,<value >,...SI5351 setup group
20:28:52: sts01,21*  21 BEGN PGM                                Key TX and Send Callsign (CW)
20:28:52: sts01,22*  22 CODE PGM       <message>                Send Message (CW) up to 22 char
20:28:52: sts01,23*  23 TALK PGM       <file-name>              Play Voiced Message (EDMP TALK)
20:28:52: sts01,24*  24 DONE PGM                                Send Callsign (CW), SK (CW), and unkey TX
20:28:52: sts01,25*  25 BATC PGM       <mod>,<key>,<setpoint>   Transmit Code Battery Report
20:28:52: sts01,26*  26 BATV PGM       <mod>,<key>              Transmit Vocal Battery Report
20:28:52:                                                       mod: "E" encode (not CW) for BATC
20:28:52:                                                       mod: "B" battery reading taken before BEGN
20:28:53:                                                       mod: "A" battery reading taken after BEGN
20:28:53:                                                       key: "V" battery voltage, "I" battery
      ↪ current, "R" 5V rail
20:28:53: sts01,27*  27 MODS SCHED     <Sname period offset>    Modulus Schedule Set
20:28:53: sts01,28*  28 MODC SCHED     <Sname>                  Modulus Schedule Clear
20:28:53: sts01,29*  29 MODW SCHED                              Modulus Wait
20:28:53: sts01,30*  30 TALK DIRECTORY esav TALK=name,Strt,Len,rate    (appears in FRAM as the TALK= file
      ↪ )
20:28:53:                                                       Waveform Directory Entry
20:28:53:                                                       rate keys: 4K 5K 8K
20:28:53: sts01,31*  31 ESAV FRAM      NAM=<text>               Save named record in next free location
20:28:53: sts01,32*  32 EDMP FRAM      "match string"           Dump active records
20:28:53: sts01,33*  33 EDID FRAM                               Flash JEDEC-ID table dump (PROG & WAVE)
20:28:53: sts01,34*  34 ERAS FRAM      <number> or "DEV"        Rewrite <record> to REM-
20:28:53: sts01,35*  35 EZER FRAM      <number>                 Erase <record> to ZERO
20:28:53: sts01,36*  36 HERA FLASH     <size <start>>           Hex erase (WAVE device)
20:28:53: sts01,37*  37 HDMP FLASH     <length <hex-start <*>>> Hex dump (WAVE device)
20:28:53: sts01,38*  38 :hex FLASH-HEX :llaaaattddddddddcc      Intel HEX loader (WAVE device)
20:28:53: sts01,39*  39 HALT TEST                               Halt Processor
20:28:53: sts01,40*  40 STOP TEST                               Stop Processor
20:28:53: sts01,41*  41 REST TEST                               Reset System
20:28:54: sts01,42*  42 TEST TEST                               Hardware Test Subsystem
20:28:54: STS01,43* Handler_HELP (cmd_help.c*)  2.59 Sec
20:28:54: RDY00,00* (Sp=0xBFA0)+1951 20:28:54.730
```

16.2 Sample STAT

Example Stat Listing.

Listing 50: FOX_ICARC_stat.txt

```
20:27:47: sts08,00* <<<---  STAT  ********  STAT  --->>>
20:27:47: sts08,01* KC0JFQ  FOX Transmitter
20:27:47: sts08,02* Software Bld:    Dec  7 2023  3 18:45:
20:27:47: sts08,03* System Time:     20:27:47.840
20:27:47: sts08,04* Epoch Offset:    18:00:00  64800
20:27:47: sts08,05* TOY Clock:       000D51A4 00 00; Osc ON, Charge Disabled
20:27:47: sts08,06* Sys Upd Flg:     Not Set
20:27:47: sts08,07* Conf Jumpers:    NOT_Master NOT_Test
20:27:47: sts08,08* Flash Prog  U3:  04.49.03 MB85RS4MT    FLASH_FRAM    Fujitsu   4096K-bits  16384-records CMD4
20:27:47: sts08,09* Flash WAVE U12:  EF.70.17  W25Q64JV    FLASH_PAGE    Winbond  65536K-bits    1000-seconds CMD4
20:27:47: sts08,10* Flash HEX Dev:   WAVE
20:27:47: sts08,11* Battery, Idle:   8.584V[0x036F]  30mA[0x003E]
20:27:47: sts08,12* Analog Others:   Reg-5V: 5.078V[0x0208]  Switch: 2.498V  CdS-Cell: 2.498V
20:27:47: sts08,13* UART buffer:     143 (NET:0, USB:66)
20:27:47: sts08,14* <<<--- Scheduling  PARAMETERS --->>>
20:27:47: sts08,15* MOD Schedule  00   Idle  S0= 300 0
20:27:47: sts08,16* MOD Schedule  01   Idle  S1= 300 60
20:27:47: sts08,17* MOD Schedule  02   Idle  S2= 300 120
20:27:47: sts08,18* MOD Schedule  03   Idle  S3= 300 180
20:27:47: sts08,19* MOD Schedule  04   Idle  S4= 300 240
20:27:47: sts08,20* <<<--- TRANSMITTER PARAMETERS --->>>
20:27:47: sts08,21* Callsign:        KC0JFQ
20:27:47: sts08,22* Nickname:        FOX20
20:27:47: sts08,23* Radio Config:    0x01C3 SI5351 T0 TX_ENA TONE PWMH0
20:27:47: sts08,24* Si5351 Config:   0x0001 CLK0
20:27:47: sts08,25* Si5351 Divisor:  0x139E,0xACA7F,0xF4240 0x0100,0x00,0x01
20:27:48: sts08,26* Frequency:       144.150 (Xtal: 20.000 MHz)
20:27:48: sts08,27* CW config:       25 WPM 1,3,7,14 [0x1D4C] 0.600KHz
20:27:48: STS08,28* Handler_STAT I (cmd_stat.c*)  0.97 Sec
20:27:48: RDY00,00* (Sp=0xBFA0)+1951 20:27:48.730
```

16.3 ICS525 20MHz Frequency Table

```
//  1064 FILE ics525_data_test.h
    //===============================================================//
    //                                                               //
    //    IDT-525-02 frequency control structure... 525              //
    //      Input Clock Frequency  20.000 MHz                        //
    //      Output Divisor   1 (Out_Div column)                      //
    //                                                               //
    //    NOTE that we calculate the frequency table using double    //
    //      precision numbers.  The zNEO will be using single        //
    //      precision, so the frequency number stored in the         //
    //      zNEO will differ in the least significant digits.        //
    //                                                               //
    //===============================================================//
#ifndef __ics525__
#define __ics525__
    struct IDT525 {
                float Frequency;        // Programmed Frequency (in MHz)
                float Offset;           // Offset from 5KHz channel center
       unsigned short VCO_Div;          // VCO Divisor (9 bits)
       unsigned char  Ref_Div;          // Reference Divisor (7 bits)
       unsigned char  Out_Div;          // Output Divisor (3 bits)
                   };                   //
                                        //
#endif
```

```
           {     144.150943,   943.000,      183,       51,      6},   // R=1000
           {     144.210526,   526.000,      129,       36,      6},   // R=1000
           {     144.285714,   714.000,      194,       54,      6},   // R=1000
           {     144.375000,     0.000,      223,       62,      6},   // R=Ideal
           {     144.390244,   244.000,      140,       39,      6},   // R=500
           {     144.545454,   454.000,      151,       42,      6},   // R=500
           {     144.615384,   384.000,      227,       63,      6},   // R=500
           {     144.680851,   851.000,      162,       45,      6},   // R=1000
           {     144.705882,   882.000,      115,       32,      6},   // R=1000
           {     144.905660,   660.000,      184,       51,      6},   // R=1000
```

17 Configuration Worksheets

This worksheet may be used to map out an operating schedule for up to 8 transmitters.
The **MOD** period is 600 and the **MOD** offset is across the bottom of the X axis.
Starting with an offset of zero, plot the ON time of the first transmitter and allow for a few seconds of gap to estimate the best time for the second transmitter to start. Continue on like this until all offsets are established or it is discovered that the message time is too long.

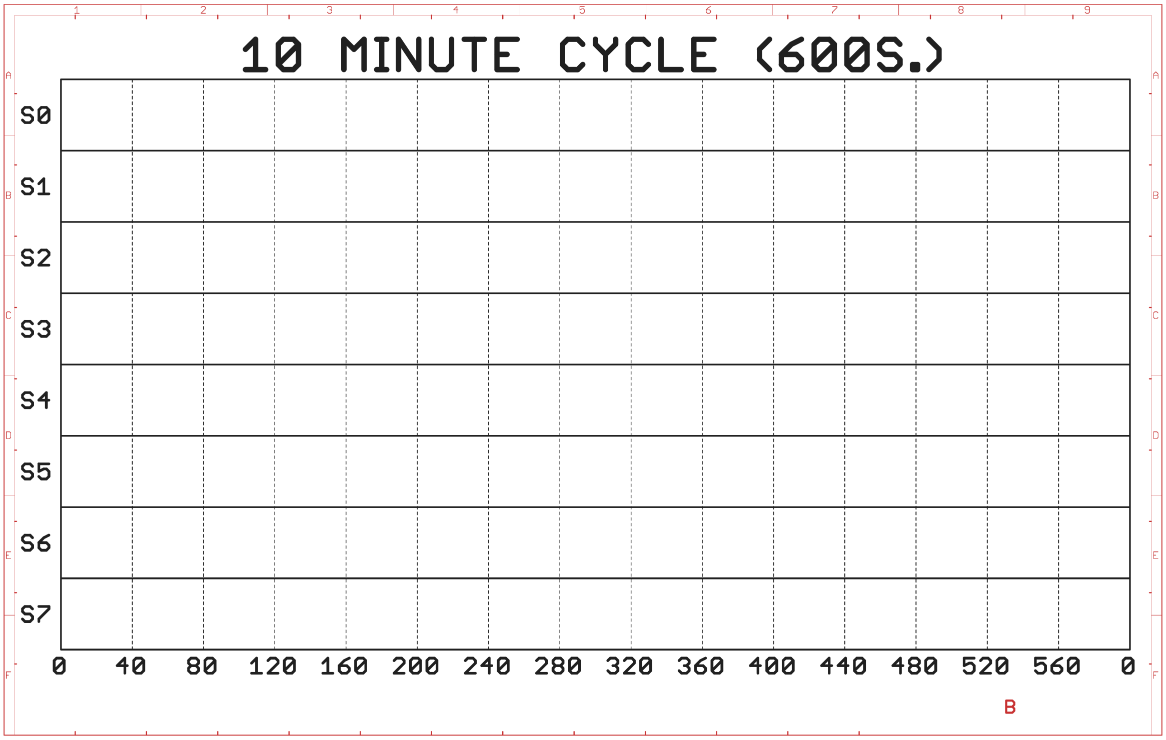


Figure 29: Worksheet, 10 minute cycle

This worksheet may be used to map out an operating schedule for up to 8 transmitters using a period of 900 seconds.

As above, start with an offset of zero, plot the ON time of the first transmitter and allow for a few seconds of gap to estimate the best time for the second transmitter to start. Continue on like this until all offsets are established or it is discovered that the message time is too long.

11767

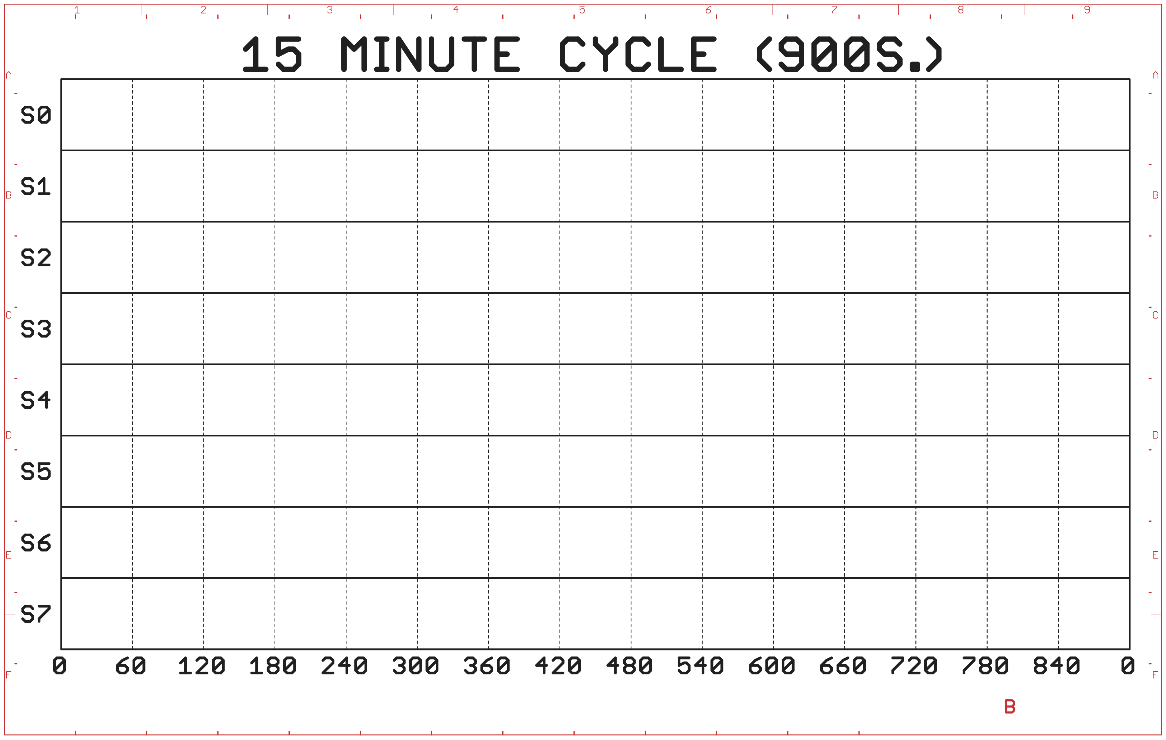


Figure 30: Worksheet, 15 minute cycle

This worksheet presents a standard 5-minute ARDF contest schedule using a set of 6 transmitters. The codewords are shown above the operating window for each transmitter.

The fox transmitters each have about a 55 second window in which to send a message. The five are time multiplexed and are all operating on the same frequency. The last transmitter, the FB line, is the finish beacon that transmits continuously on a different frequency.

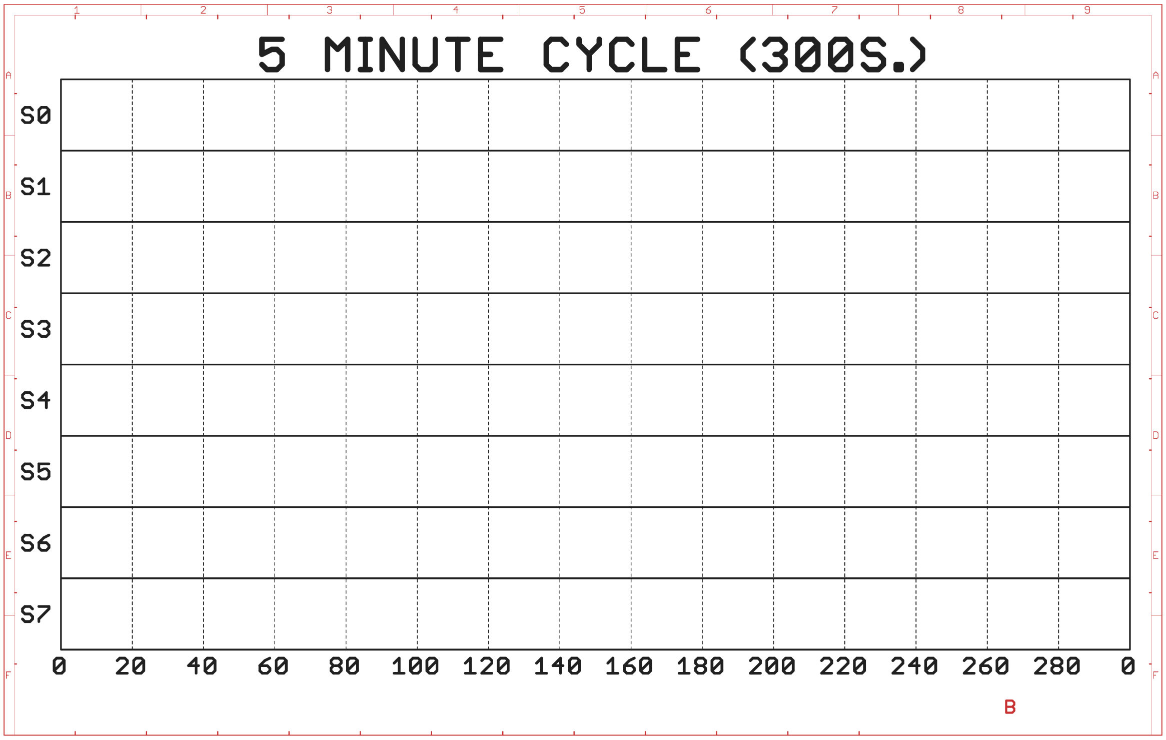
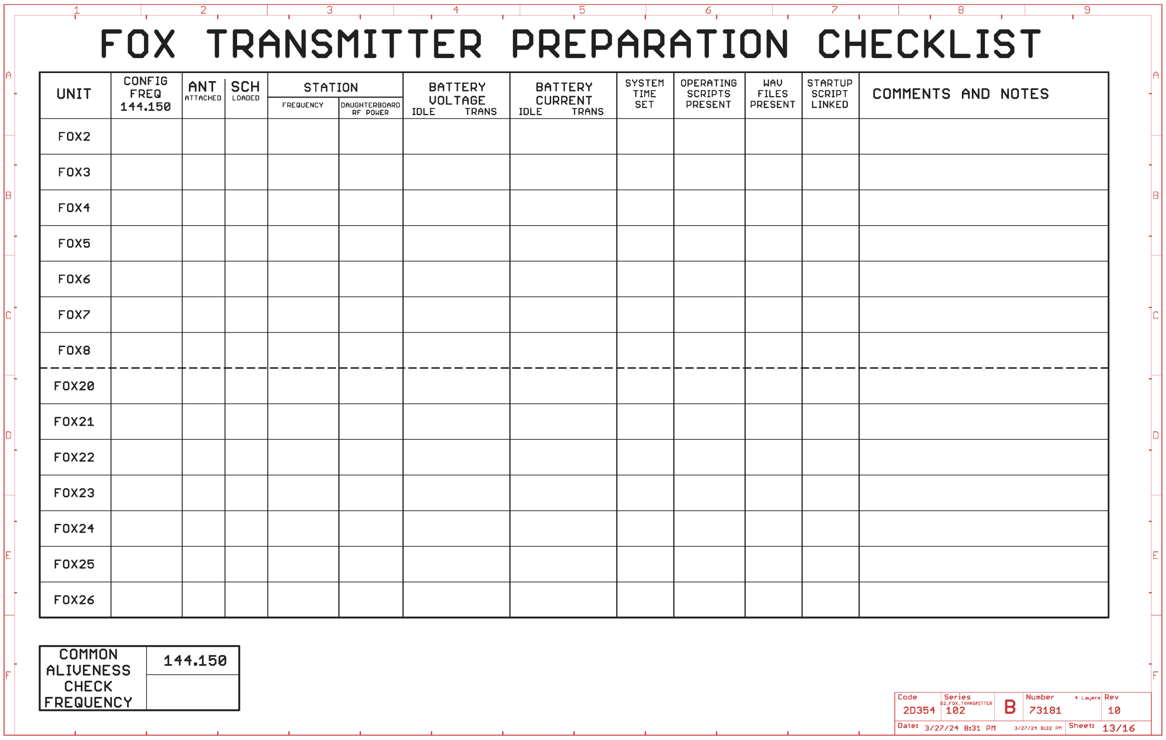


Figure 31: Worksheet, 5 minute cycle

Use these checklists to prepare for a fox hunt. As shown, the unit names and hardware notes are for the authors collection of transmitters. You can choose to use the same set of unit names to keep life simple. The authors stable of transmitters has all hardware versions while you would expect to have all the same version transmitter.



FOX TRANSMITTER PREPARATION CHECKLIST

UNIT	CONFIG FREQ 144.150	ANT ATTACHED	SCH LOADED	STATION		BATTERY VOLTAGE		BATTERY CURRENT		SYSTEM TIME SET	OPERATING SCRIPTS PRESENT	WAV FILES PRESENT	STARTUP SCRIPT LINKED	COMMENTS AND NOTES
				FREQUENCY	DAUGHTERBOARD RF POWER	IDLE	TRANS	IDLE	TRANS					
FOX2														
FOX3														
FOX4														
FOX5														
FOX6														
FOX7														
FOX8														
FOX20														
FOX21														
FOX22														
FOX23														
FOX24														
FOX25														
FOX26														

COMMON ALIVENESS CHECK FREQUENCY	144.150

Code	Series G2_FOX_TRANSMITTER		Number 4 Layers	Rev
2D354	102	B	73181	10
Date: 3/27/24 8:31 PM	3/27/24 8:32 PM		Sheet:	13/16

Figure 32: Worksheet, preparation checklist

Build your transmitter plan noting which transmitters will be in use. The *IN USE* column may be used to determine which stations will be used for the hunt. Your antenna management plan may benefit from the next column, the *ANT ATTACHED* column, to make sure you have counted antennas correctly.

Note in the *SCH LOADED* column, the names(s) of the active schedules. These will show up in the **RUN0** commands in the **INI=** file.

The station frequency should be tracked in the *STATION FREQUENCY* column. This should come from the active schedules (not from the **INI=** file). Typically the **INI=** file will immediately produce an aliveness report on a dedicated frequency.

Log the battery voltage in the *BATTERY VOLTAGE* column by turning the unit on and recording the voltage and current reports. This would typically occur the night before, so double check that power is off after the prior evening setup.

Check off the *SYSTEM TIME SET* column as the units time is updated.

The next three columns, the *OPERATING SCRIPTS PRESENT*, *WAV FILES PRESENT* and *STARTUP SCRIPT LINKED* provide logging space for setup that should only need to be done when the station is initially setup. Use the *STARTUP SCRIPT LINKED* to indicate which schedules are active in this station.

The last column may be used to record the active schedule to verify that a sane set of schedules are loaded into the transmitters.

FW Kent Park FOX HUNT

Transmitter Time Synchronization Checklist

Tue Mar 26 21:55:15 2024

Event Callsign: **W0JV** Start: End:

11865

Name	Chk Freq Oper Freq	Drwg Nmbr Daughterboard	Power S/W Ver	Batt Voltage	Batt Current	Event Validation Code Event ID Code	Test Jumper
FOX2 Unit: 1	144.305MHz 144.305MHz	102_73161_12	12mW V2.03	Idle Tx	No Sensor	144.305 / **7KGOOQ9D** ID:**291349**	Verify NOT INSTALLED
FOX3 Unit: 2	144.335MHz 144.335MHz	102_73161_12	12mW V1.52	Idle Tx	No Sensor	144.335 / **U29LS6PK** ID:**264196**	Verify NOT INSTALLED
FOX4 Unit: 3	144.565MHz 144.565MHz	102_73161_12	5mW V1.52	Idle Tx	No Sensor	144.565 / **O5Y9R5A0** ID:**356476**	Verify NOT INSTALLED
FOX5 Unit: 4	144.150MHz 144.285MHz	102_73161_25 102_73161_24	45mW V3.54	8.30V Idle Tx 8.20V	No Sensor	144.150 / **NHGMG1J6** ID:**683174**	Verify NOT INSTALLED
FOX6 Unit: 5	144.150MHz 144.285MHz	102_73161_25 102_73161_24	45mW V3.54	8.10V Idle Tx 7.90V	No Sensor	144.150 / **3952YFTQ** ID:**112749**	Verify NOT INSTALLED
FOX7 Unit: 6	144.150MHz 144.285MHz	102_73161_25 102_73161_24	5mW V3.54	8.70V Idle Tx 8.60V	No Sensor	144.150 / **6PLGUGN2** ID:**610728**	Verify NOT INSTALLED
FOX8 Unit: 7	144.150MHz 144.285MHz	102_73161_25 102_73161_24	5mW V3.54	8.40V Idle Tx 8.30V	No Sensor	144.150 / **AZ6PLH9P** ID:**879185**	Verify NOT INSTALLED
FOX20 Unit: 8	144.150MHz 144.285MHz	102_73181_5 102_73161_28	50mW V3.54	8.90V Idle Tx 8.70V	31mA Idle Tx	144.150 / **6QQASSLG** ID:**627907**	Verify NOT INSTALLED
FOX21 Unit: 9	144.150MHz 144.225MHz	102_73181_10 102_73181_36	110mW V3.54	9.00V Idle Tx 8.70V	25mA Idle Tx 215mA	144.150 / **8METGOUX** ID:**145441**	Verify NOT INSTALLED
FOX22 Unit: 10	144.150MHz 144.225MHz	102_73181_10 102_73181_36	110mW V3.54	9.30V Idle Tx 9.10V	42mA Idle Tx 208mA	144.150 / **B09WU0F8** ID:**420663**	Verify NOT INSTALLED
FOX23 Unit: 11	144.150MHz 144.225MHz	102_73181_10 102_73181_36	140mW V3.54	8.90V Idle Tx 8.50V	56mA Idle Tx 338mA	144.150 / **2Y1XAR5A** ID:**243144**	Verify NOT INSTALLED
FOX24 Unit: 12	144.150MHz 144.225MHz	102_73181_10 102_73181_36	100mW V3.54	9.20V Idle Tx 9.00V	51mA Idle Tx 310mA	144.150 / **A8HHR1QU** ID:**220033**	Verify NOT INSTALLED
FOX25 Unit: 13	144.150MHz 144.225MHz	102_73181_10 102_73181_36	120mW V3.54	9.00V Idle Tx 8.60V	42mA Idle Tx 317mA	144.150 / **NF51ZNDM** ID:**905997**	Verify NOT INSTALLED
FOX26 Unit: 14	144.150MHz 144.225MHz	102_73181_10 102_73181_36	140mW V3.54	8.10V Idle Tx 7.90V	48mA Idle Tx 337mA	144.150 / **MR86Q9CB** ID:**066314**	Verify NOT INSTALLED

Print "fox_label_A_bot.ps" label file. Attach labels when the TOY clock is updated.

Some units may not report transmit current correctly. Only 102-73181-10 units verbalize voltage and current reports.