ICARC Fox Transmitters

William Robison

July 1, 2025

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/HamRDF

LaTeX Source Files:

Contents

List of Figures

List of Tables

Chapter 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 Chirp

A transmission method (not a RADAR chirp) where the transmitter enables carrier for a short period, in effect emulating a wildlife tracker.

1.2 Chip & Chipping

The word **Chip** refers to the smallest unit of CW activity managed by the transmitter.
A *dit*, being the smallest CW unit, takes one chirp time to send.

We use **Chiping** when talking about the assembly and delivery of a CW message that is built on a list of *chips*.

1.3 FLASH

An in-circuit erasable and programmable memory.

A type of non-volatile memory that exhibits very asymmetric access speed. Write speed is several orders of magnitude slower that read speed.

Update is handled by erasing the entire device and then loading it one (32 byte) record at a time.

1.4 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. The type of memory is byte accessible (for both read and write) as well as non-volatile.

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

1.5 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.

1.6 MMIC

Monolithic Microwave Integrated Circuit.

Amplifier on a chip that makes life simple at VHF frequencies.

1.7 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.

1.8 Program

This refers to the code in the zNEO system-on-chip.

1.9 Sequence

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

Typically this *sequence* is stored in external FRAM memory.

1.10 TOY Clock

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

1.11 xxx

xxx (the $24^{\text{th.}}$ letter of the alphabet)

Chapter 2

Motivation

Why all the improvements?

Because we can! And because if FUN! (Parts availability forced the 102-73181-10 update this time around).

Most of the shortcomings of the 102-73161 series boards are addressed in this revision, at least all the known shortcomings. The clock synthesizers used in the earlier 102-73161 boards are all at end-of-life and will become more difficult (i.e. impossible) 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, allowing operation in the UHF band.

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.

> This requirement/responsibility may be imposed on the hunt operator.

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

> This implies code or voice.

2.1.2 Multiple frequency

We must be able to (conveniently) operate the transmitter on more than one frequency.

> The SI5351 provides operation throughout the 2M band.

2.1.3 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.

Using a 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.

The use of a low power RF power amplifier extends battery live.

> A 100mW output level will provide coverage adequate for an area such as a city or county park.

2.1.4 Code Storage

We must store enough bits to identify the transmitter.

> An external serial memory dievice.

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

> This same external memory device.

2.1.5 Physical Size

The housing in which the transmitter lives must be a manageable size.

> Too small, and it's hard to find. We need to set out flags or something similar to make the device such that it can physically be seen.
>
> Too large, and it's hard to transport. It shouldn't be so large that you can't carry a full hunt group (5 to 10 units) during setup and teardown.

2.1.6 Programmable without special tools

It is essential that the end user is able to load the operating instructions (FRAM) as well as the audio waveforms (FLASH) without the need for special tools.

> Serial access allowing a simple terminal program to be used to load the memory devices. A simple control program can be made to automate this task.
>
> We may also implement a fast binary loader that operates through this serial channel to spped up loading operatons.

2.2 Desirements

These are desired features

2.2.1 Dynamic frequency

We would like to be able to easily change frequencies during operation.

> As a convenience for the hunt organizer, we want to have the transmitter emit a *wellness/alive message* on a common frequency when powered on.
>
> The transmitter will then switch to its assigned operating frequency for the hunt.

This capability leaves the door open to some very devious operating possibilities.

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.
>
> The 102-73181-10 and later boards provide for operating with an RF module eliminating the SI5351 altogether. This opens up the possibility of operating in the UHF band.
>
> The 102-73181-5 and 102-73181-10 boards both can operate in the 6M band. The SI5351 is easily programmed to generate carrier in the frequency range.
>
> We may also employ an RF module on a daughterboard to cover the UHF band (i.e. SA818/DRA818).

2.2.3 Operation on 80M

We would really enjoy being able to operate on the 80M band. This allows conducting an **IARU** event.

> Can we pull this off without having to alter a Fox Transmitter that is also used on the 2M band?
>
> It looks like we can fit a simple HF amplifier with a low pass filter on an RF daughtercard. See the details in section 4.17.8 on page 92.
>
> Although aimed at the SI5351, the ICS525 should be able to hit some frequencies in the 80M band.

2.2.4 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).

A 64Kb device holds 256 command/directory records.
A 256Kb device holds 1024 command/directory records and is under $5.00.

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.

2.2.5 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 of 5KHz rate.

> In the 102-73181 design, as noted above in section 2.2.4, 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.

2.2.6 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.

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.

2.2.7 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.

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.

2.2.8 Large memory footprints

A large program flash in the zNEO allows the implementation of a comprehensive control language.

A large FRAM allowing for the storage of audio waveform data. Large devices open the door to having a verbal hunt where message traffic doesn't repeat.

2.3 Scheduling Philosophy

Throughout the software there has been an effort to be disciplined in how the schedules (at several levels) are implemented.
The scheduling method is descripbed in section 9.1 on page 137.

This now (as of V3.75) extends down to the way timing is controlled when emulating a wildlife tracker.

The general scheduling is synchronous with the TOY clock (through the use of modular arithmetic) to allow multiple units to operate using the same frequency without stepping on top of each other.

2.4 Fast Loading

One very late arriving goal was to be able to load the FLASH memory in a reasonable time. There was never a seperate InTel HEX loader utility, rather we were using the *fox_simple* utility (see section 12 on page 235) to pass a HEX file through to the target system. The *fox_simple* utility runs open-loop and simply uses a fixed delay between lines of text sent to the targeet. Loading a waveform image that holds several hundred kilobytes of image data is slow, at best.

There was a growing desire to improve the situation as audio file loading was becoming a long drawn-out affair. A proposed binary protocol was implemented to reduce the communications bandwidth requirements and operate closed loop to minimize dead time.

See discussion of the *fox_binary* utility , in section 12 on page 235, for a discussion and description of how this works.

Chapter 3

Revision History

List of updates to the hardware and software.

3.1 Software

Updates and changes to the Operating Software

3.1.1 V4.11

Looking at the older boards, an EEPROM device looks to be an attractive way ($$$) to provide a large audio file system.

So three ST Micro devices, from 8Mb to 32Mb, have been added to the memory device table.

We also add a marker for EEPROM that tells the system to treat it the same as a FLASH device. It should handle an overwrite just like FRAM...

3.1.2 V4.10

Eliminate almost **all** frequency limits checking. This feature is mostly meaningless as you should be placing **only** valid frequency settings in the external table.

This should make working with the 102-73227-11 RF amp a bit easier. This is an HF amplifier with provisions to accommodate the ICS525 frequency synthesizer that operates above 30MHz. The 102-73227-11 RF amp has a *divide by 16* between the synthesizer and the RF amplifier.

Eliminate almost **all** internal frequency tables.

The SI5351 table has **only** 144.100 MHz to allow a quick characterization.

The ICS525 has **all** frequency entries externalized. This means this version is compatible with all hardware versions!

3.1.3 V4.09

Start of support for 80M RF daughterboard amplifier.

Change ICS525 support to require external frequency table. This may still fit in both hardware revisions to allow for a single load image for all!

Some of the **D525** sub-commands eliminated, specifically there is no ICS525 setup table to dump.

3.1.4 V4.08

ARRGH!!!
All this time the **<CALL>** and **<NAME>** substitutions were not implemented for the code generator.

Also added an alias of **<NICK>** to more closely match the command verbs.

You will need to be up at this revision to run and IARU sequences that don't talk...

3.1.5 V4.07

Update *cmd_frequency.c* to allow selections in the FM broadcast band.

This requires an entry in the external frequency table. You will also have to measure the carrier frequency and adjust the SI5351 register values as needed.

3.1.6 V4.06

Add the **TIRP** command to vebalize system seconds. Allows you to see how close to correct time the fox transmitter clock is running using only a radio.

This adds a need for two new utterance entries: *V_SEC* and *V_TIRP*. The first should say "seconds" and the second should say "time sync".

Removed an instruction in the block loader ISR.

Extra status read is out (removed).

Three logical timer channels have been removed (reduce RTI ISR overhead)).

3.1.7 V4.05(not working!)

Pushing to shrink the ISR's as much as possible.

Managed to remove a couple of lines in the UART ISR...

An unused status register read, and a couple of places that force a zero into the input buffer after saving the current character. Not necessary for the binary mode.

Also reduce the path length in the timer ISR.

Remove three logical timer channels that were unused and being checked in the timer ISR.

It looks like the timer and UART times combined are less than one character time at 115,200. Can't shut the RTI timer off as it's used in the binary loader to imiplement delays when the protocol is active.

Running at 230,400 might be possible if it weren't for the RTI interrupt. The UART in the zNEO is **not** buffered (I miss the SIO/SCC from Z80 days), so we can't deal with any higher data rates unless we fire up the DMA controller.

3.1.8 V4.04

ARRGH!

Handling of **CONF CW** and **CONF FM** was a little off. It should return to continuous carrier after we send **CONF FM**.

SO, in cmd_voice and cmd_code always enable the **TX_ENA** net before sending. The code then tests the *TX_ENA_MASK* bit in the configuration flags to shut down the **TX_ENA** net after.

Note that **CONF CW** will operate in what sounds like full break-in mode where carrier is off between dit/dah.

The **CONF FM** command does **not** directly manipulate the **TX_ENA** net, so we have to have voice or code traffic to transition into that mode.

You must also have the 102-73181-28 or the 102-73181-71 RF amplifier installed for this to work correctly!

Added several 8M, 16M and 32M devices from *BYTe Semiconductor* and *Renesas*. Trying to get the FLASH table generously populated with narrow package devices (look for 0.154" wide package)

Note that both **FLASH_NVRAM** and **FLASH_AAI** have been deprecated. The SPI handler no longer recgonizes them.

A later addition to V4.04 improves performance of the UART ISR slightly. This is to allow operation at 115,200 b/S to slightly improve load times.

The ISR now checks the UART1 status register near the end of processing and reads a 2nd. character if it is available. This aims to reduce or eliminate overrun errors where there is only 85 to 95 between characters.

Some of the debug code has also been removed from the ISR to further reduce path length.

3.1.9 V4.03

Was (all of a sudden) having problems with **ONCE** command. Add line to strip off all trailing space and control characters.

3.1.10 V4.02

Count up NAK sent out as well as record count. Pass it back in the status report at end of activity.

Also noticed that AAI (Auto Address Increment) FLASH devices don't work right, so removed 3 AAI entries from the device table. These devices are deprecated.

If they appear on a board, they will need to be replaced!

3.1.11 V4.01

Arrgh! We should be able to do this for both external memory devices. And now we can.

Change the **B** modifier to **PROG** and **WAVE**. The modifier causes the switch to binary mode and selects the target device.

3.1.12 V4.00

A rather major upgrade to drastically improve the speed with which we can load waveforms into FLASH memory.

Modify both **H115** and **H56K** commands to invoke the binary load protocol by adding a **B** modifier.

3.1.13 V3.95

A bit more tidy-ing up...

Add **CONF XTAL nn.nn** to better document when we're running the SI5351 with a non-standard crystal.

In the 102-73181-10 world, the more-or-less standard crystal is 20.0MHz, the same one used by the zNEO (to minimize unique parts).

3.1.14 V3.94

No chance to keep CW and -AM straight, so add decode for **CONF FM** to make command sequences a bit clearer.

This decodes a bit funny

3.1.15 V3.93

Changed the bit rate divisor for 115200 operation. Prior revisions probably won't have a working **H115** command.

3.1.16 V3.92

Changed the limits for 2M, 6M, 10M, and 20M operation. Added limits for 70cM (SA818U)

- 70cM 432.00MHz to 435.00MHz
- 2M 144.090MHz to 147.990MHz
- 6M 50.010MHz to 53.950MHz
- 10M 28.150MHz to 28.300MHz
- 20M 14.095MHz to 14.230MHz

No practical effect on existing units.

3.1.17 V3.91

Add "I" flag to BATR command to dump coefficients table.

3.1.18 V3.90

Add timetag to BATR command so we don't depend on external timetag function.
This just pulls the currrent time and dumps it in the BATR report.

3.1.19 V3.89

No functional changes.
Cleaned up some text reports (eliminate redundant text).

3.1.20 V3.88

Thought there was a problem in the **STAR** command...
There wasn't but did notice that the time field needs to be complete (i.e. HH:MM:SS).
Rearranged the reporting line in the **STAT** command.

3.1.21 V3.87

Add battery voltage limit test to the **BATV** command. It operates just like the **BATC** command, sending out *SOS SOS* in code following the voltage vebalization.

3.1.22 V3.86

Move the "*TALK*" test from cmd_message to cmd_voice.

No change to the way things operate.

3.1.23 V3.85

Ran into to peculier operation when embeding **REM-** in a sequence so updated the *cmd_help* module to clear the first character after the *REM-* to zero to stop further processing on the line.

3.1.24 V3.84

Alter the CHRP command to process TALK files.

CHRP FILE period offset delay count.

3.1.25 V3.82

Alter the way the the ERAS and ESAV commands deal with record erase.

Erase record (ERAS block-number) rewrites the record with the text string **MT****. Save record (ESAV text-to-save) will overwrite an empty record or thge **MT**** record.

You will see the **MT**** record when dumping (EDMP).

Makes sanity checking a bit easier.

3.1.26 V3.81

Add the schedule name to the RUN0 status report.

A bit of a sanity check when looking at command traffic during sequenct development and debugging.

3.1.27 V3.80

Add synchronous feature to WAIT command.

You can giv a period/pffset style of argument to **WAIT** to invoke synchronous scheduling.

We also update **EZER** and **ERAS** to accept a range of record numbers to get rid of a group all-in-one command.

3.1.28 V3.77

Add a frequency parameter to the **BEGN SILENT** command to enable the tone generator at the specified frequency.

This addition is to aid in filter performance analysis.

The SI5351 frequency test table is also updated in this version. The table has entries at 500KHz intervals from 144.100 to 147.500 to allow testing across the entire band.

3.1.29 V3.76

Update the **TIME** command so that a time set operation is synchronous.

When using a **TIME** command with no arguments (to set the system time from the TOY clock), the TOY clock is read until the LSB changes. This gets the fox transmitter system time to within 10mSec of the TOY clock.

3.1.30 V3.75

Fox_73181_3.75_2024-11-03T11.tar.gz

Modify the **CHRP** command to operate synchronously.

See the command reference table 10.36 on page 196.

Backed out the V3.74 change to send nickname, it seems to overflow a data buffer.

The *command.c* module holds the help text. Early V3.75 release do not have the updated help text. Version 3.22 of the *command.c* module has update help text string.
No change to code (hence no change to V3.75 string).

3.1.31 V3.74

Fox_73181_3.74_2024-11-01T15.tar.gz

The **BEGN** command now sends the nickname as part of the signon CW traffic.

3.1.32 V3.73

Fox_73181_3.73_2024-08-19T15.tar.gz

Add the **BATR** command to track battery voltage.

Use this in the sequence to leave a set of battery readings for later battery capacity analysis.

This command was added to produce the plots in section 4.10.2 on page 61.

3.1.33 V3.72

Fox_73181_3.72_2024-07-09T10.tar.gz

Add the **FOFF** command to track the offset we have applied to the frequency setup table.

3.1.34 V3.71

Fox_73181_3.71_2024-07-08T12.tar.gz

Change default frequency table for SI5351 to **0**KHz and leave it that.

User then has to measure the offset and load an external frequency table.

This way the base image for the zNEO is identical on all units.

3.1.35 V3.70

Fox_73181_3.70_2024-06-16T19.tar.gz

Change default cap on the reference crystal to 8pF. User can move up or down from there if needed. Also change the internal frequency table.

The **cmd_stat.c** module didn't quite get the changes to the *SI5351_clk_ena_bits* field right.

Copied decode for this field from the **cmd_conf.c** module.

3.1.36 V3.69

Fox_73181_3.69_2024-06-11T15.tar.gz

Changed control terminal bit rate to 57,600. Don't really see any reason not to run at that rate all the time.

Removed H56K and replaced it with H115.

3.1.37 V3.68

Fox_73181_3.68_2024-06-11T15.tar.gz

Added the nPF sub-commands to the **CONF** command.

Now we might revisit the frequency offset used in the SI5351 register configuration utility and change the default load capacitance...

3.1.38 V3.67

Fox_73181_3.67_2024-06-10T14.tar.gz

The HERA command deals with block erase now.

See command description table, figure 10.55 on page 211.

3.1.39 V3.66

Fox_73181_3.66_2024-06-10T12.tar.gz

The STAR command (start scheduling after specified time) appears to work now.

There is a common time compare routine used by the scheduler and the STAT command.

See command description table, figure 10.13 on page 171.

3.1.40 V3.65

Fox_73181_3.65_2024-05-28T08.tar.gz

Add the **CONF AM/CONF CW** configuration bit.

This sets the transmitter to operate in an interrupted carrier mode of operatioon.

See command description table, figure 10.16 on page 172.

3.1.41 V3.64

Fox_73181_3.64_2024-05-24T10.tar.gz

Changed the default T2 time in cmd_conf.c from 50mS to 150mS. This should give the RF generator time to start before the CW signon message starts .

3.1.42 V3.63

Fox_73181_3.63_2024-05-22T19.tar.gz

Fixed processing of RUN0 argument.

Should match sequence name, not use the number as an index into the schedule table which may be sparsely populated.

This revision is more-or-less required for correct operation with a sparsely populated scheduleing table.

3.1.43 V3.62

Fox_73181_3.62_2024-05-09T16.tar.gz

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.

3.1.44 V3.59

Fox_73181_3.59_2024-04-05T15.tar.gz

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 after 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*).

3.1.45 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...

3.1.46 V3.56

Fox_73181_3.56_2024-04-09T18.tar.gz

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>

3.1.47 V3.54

Fox_73181_3.54_2024-03-29T19.tar.gz

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).

3.1.48 V3.53

Fox_73181_3.53_2024-03-22T12.tar.gz

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

3.1.49 V3.52

Fox_73181_3.52_2024-03-21T18.tar.gz

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...

3.1.50 V3.51

Fox_73181_3.51_2024-03-21T16.tar.gz

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

3.1.51 V3.50

Fox_73181_3.50_2024-03-20T14.tar.gz

Update to implement the **CHRP** command.

Emulate a wildlife tracker.

3.1.52 V3.31

Fox_73181_3.31_2024-03-12T20.tar.gz

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).

3.1.53 V3.28

Fox_73181_3.28_2024-03-06T20.tar.gz

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.

3.1.54 V3.30

Fox_73181_3.30_2024-03-12T18.tar.gz

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.

3.1.55 V3.27

Fox_73181_3.27_2024-03-04T10.tar.gz

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.

3.1.56 V3.26

Fox_73181_3.26_2024-02-20T14.tar.gz

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

3.1.57 V3.24

Fox_73181_3.24_2024-02-14T18.tar.gz

First test on the 102-73181-10 hardware.

Update FLASH tables.
Update GPIO_MASTER.

3.1.58 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.

3.1.59 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.

3.1.60 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.

3.1.61 V3.16

Release Date Dec 2023.

Condense the battery reporting into the BATC and BATV commands.

3.1.62 V3.15

Release Date Dec 2023.

Implement the ONCE command.

3.1.63 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.

3.1.64 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.

3.1.65 V3.10

Release Date 12 Nov 2023.

Merging in the SI5351 management code.

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

3.1.66 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*.

3.1.67 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.

3.1.68 V2.00/2.01

Release Date 10 Jun 2020.

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

3.2 Hardware Revisions

Artwork and component selection notes.

3.2.1 102-73181-10

The zNEO seems to be impossible to get in the 80-pin package. This reworks the circuit board to use the 64 pin package.

This affects the **MASTER** signal. Software accommodates the different port pins that the **MAS** signal connects to.

Repurposed the (physical) network time port. It wasn't being used in the field and the DRA818 module will make better use of 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. This is a surface mount part.

The SI5351 can directly drive the RF daughter board with one of its clock pins (good for about 5mW).

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

Improve the charge circuit that keeps the lithium coin cell topped up.

This revision is **required** when using the DRA818/SA818 daughterboards.

3.2.2 102-73181-5

This hardware does not support the DRA818/SA818.

Switch to yet another clock generator, the SI5351.

Add charge circuit to keep lithium coin cell topped up.

This revision is **not** compatioble with the DRA818/SA818 daughterboards.

3.2.3 102-73181-0

This hardware does not support the DRA818/SA818.

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 is **not** compatioble with the DRA818/SA818 daughterboards.

3.2.4 102-73161-25

This is the last revision of the original series of boards that all used the ICS525 clock synthesizer. This revision is **not** compatioble with the DRA818/SA818 daughterboards.

Chapter 4

Theory of Operation

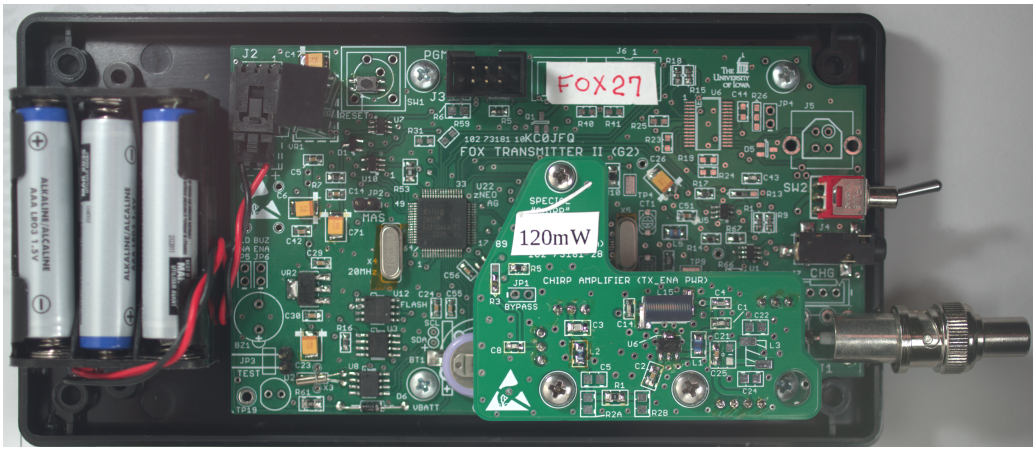


Figure 4.1: 3rd. generation transmitter (102-73181-10)

The 102-73181-10 unit incorporates all the good features of the previous units. The 102-73161-25 and the early 102-73181 boards are similar enough to allow the 102-73181-10 software to operate all of them.
The 102-73161-7 and 102-73161-12 boards are dissimilar enough (and exist in such small numbers) so as not to be compatible.

Note the 3.5mm jack, center right. This provides for a lower cost serial connection to load the operating sequences into the transmitter (this applies only to the 102-73181-10 revision). The serial command jack on the 102-73181-5 artwork (a vertical 3.5mm connector colocated with J5) 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 module (DRA818/SA818 or handy talkie) and moves the control port function to the external 3.5mm jack.

Although the USB UART pads are retained on the board, the FT232 would not be normally be populated (this is to reduce cost and improve useability). The *more-or-less* nominal configuration is shown in the image. Take note that the 3.5mm (host control) jack is accessible without opening the enclosure (streamlining setup on the night-before).

These latest iterations (102-73181-5 and 102-73181-10) add a circuit to keep the (on-board) clock 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 as only 1 signal was affected by the smaller package.

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 a dedicated pin (to allow for reliable functioning of the DRA818/SA818 daughter board).

4.1 Block Diagram

System block diagram:

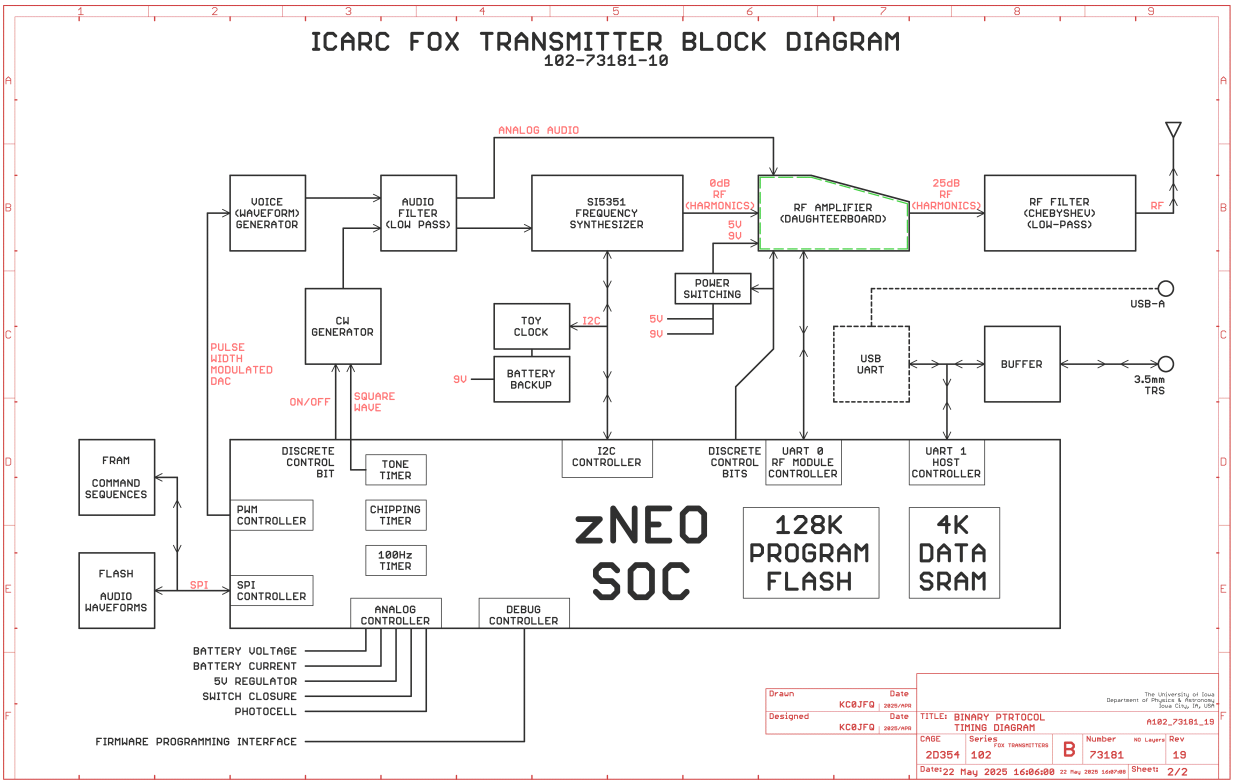


Figure 4.2: Block Diagram

As indicated at the top of the diagram, this is specifically for the 102-73181-10 revision.

Earlier revisions have a different RF synthesizer and there are fewer control signals to the RF daughterboard.

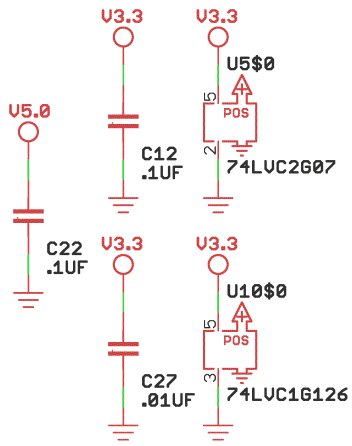
4.2 Schematics & Circuit Boards

Drawing philosophy for schematics and boards. A discussion of why the schematic and assembly drawings look the way they do.

The design flow makes use of **Eagle 6.5**. Much of the schematics appearance is directly attributable to the way **Eagle 6.5** manages a design project.

4.2.1 Power & Ground

In general, ground connections are depicted with a ground symbol immediately adjacent to the part that requires the ground connection. In a similar manner power connections are depicted with a power symbol immediately adjacent to the part that requires the power connection.

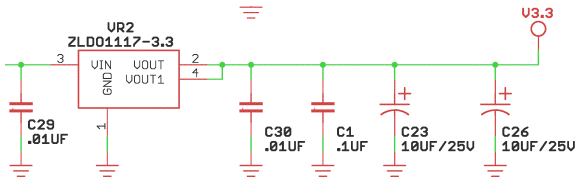


The power symbols all look similar, a circle with the net name immediately above. The power net, then, should be obvious from the net name associated with the (circular) symbol.

Net lines to connect various nodes to ground are avoided. There is an attempt to keep ground net lines to under one inch (when viewed at full size).

This is an attempt to make connections obvious, to avoide having to chase a signal net to find where it terminates or connects.

There are many instances where a cluster of bypass capacitors are connected to a longer power net *line*. This attempts to indicate rough placement of bypass parts.



This example shows bypass parts adjacent to a linear regulator. When reviewing the bypass requirements on the *ZLDO1117-3.3* the schematic is attempting to quickly show that bypass requirements have been met.

4.2.2 Net Connections

Signals that flow off-sheet use a '*Label*' with '*Xref on*' (Eagle terms). This produces a rectangle visual with a pointed side. Inside the visual is the signal name (i.e. the net name) and a sheet reference. The sheet reference is composed of a sheet number and a grid location within the sheet.

An example text visual (this example take from sheet 3 of the Fox Transmitter schematic) will look like this:

Where **VCMO_TONE** is the net name **/4** is the sheet where the signal flows and **.1D** is the coordinate on the schematic. The coordinates within the sheet are seen along the top, left, and right edge of the drawing.

Some nets that stay within one sheet may be labeled (**PWMH0**) so they can be referred to in external documentation (such as this manual).

4.2.3 Schematic Symbols

Basic logic symbols, capacitors, resistors, and inductors follow the U.S. style. Polarized capacitors have a slightly different look compard to unpolarized parts. Resistors are zig-zag (rather than a non-descript rectangle).

Parts are shown in a form that is appropriate for the schematic. There is not an effort to represet most parts as they appear on the circuit board.

Netlist verification is a job for the software package.

Small active parts

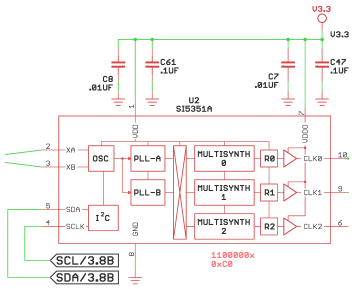
For most small active parts, the power pins are shown as seperate graphics to move the power connections away from the fuction of the gate (gate referring here to both digital and analog functions). If at all possible, the power connection visual is close to the associated gate.

The tri-state buffer in figure **??** on page 26 has power depicted in figure 4.2.1 on page 25.

Unusual parts

Unusual parts have parts symbols that expose, as much as possible, the function of the part.

A good example of this is the SI5351 on sheet 4 of the Fox Transmitter schematic. This symbol mimics that used in the manufacturers datasheet.



High pin count parts

High pin count parts, such as the processor, are typically broken into multiple synbols. The zNEO processor being a prime example.

The power pins (along with the oscillator pins and programming pins) appear as one symbol on sheet 3. The remaining pins show up on multiple seperate symbols on sheets 3 and 4.

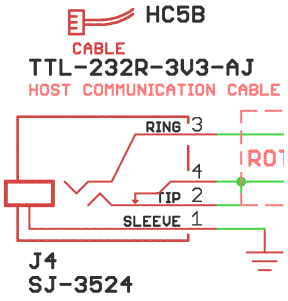
The zNEO has multiple 8 bit ports that may hve specific functions configured at run time. The zNEO package used in the design has 8 GPIO ports, each with its own (unique) symbol. The schematic symbol for these port pins are named using their GPIO assignments (i.e. PA0, PA1, PA2, etc.). Listed in the symbol next to the assigned name are the alternate function names.

The intent being a quick check that alternate function selections have been correctly routed and to provide quick reference during software development.

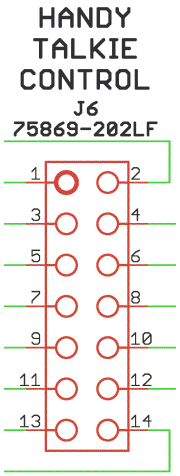
Connectors

Connectors have symbols that somewhat mimic the physical appearance of the part. In this case we are trying to make the connections evident so they can be more easily understood.

The 3.5mm TRS connector (sheet 3.1C) mimics the mechanical drawing in the datasheet.



Basic 0.025" headers are shown as simple rectangles with associated pins. In this caase the pins on the schematics are ordered to match the connector. This to avoid having a large universe of schematic symbols for any one specific connector.



A mechanical match also attempts to make clear the pin assignments for the connector (is the dual row connector numbered row major or column major?).

Worksheets

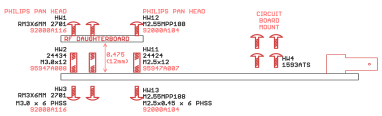
The schematics also tend to have excerpts from worksheets used in the design and in the selection of parts. A convenient example being the parts selection table for the battery monitor circuit on sheet 2.3F where parts can be selected based on the battery voltage that will be used with a specific transmitter.

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

These are text notations without any net connections. Lacking any net connection or associated physical part, these tables show on the schematic but are not transfered to the circuit board.

Mechanical Parts

Mechanical parts, such as the enclosure, spacers, and fasteners appear in the parts list. Entries for these parts are generated by adding them into the schematic (which adds a visual into the board) or directly into the board drawing.



There is a preference to have them appear in the schematic simply to have everything complete and consistent. In many drawings there are multiple instances of a particular part in order to get correct quantity into the parts list. Othere drawings amy include only one instance simply to make a note of the part number.

4.2.4 Parts Lists

The postprocessing scripts produce several formatted parts list.
Here is a brief rundown of those you may find useful.

102_73181_*.html

Web browsable parts list with same value parts grouped. There are links to DigiKey and Mouser parts.

The multiple columns are used to diagnose problems with the information stored in the schematic as well as the *Master Parts Lists* that match a part value, the package on the circuit board, and the exact part available from DigiKey.

102_73181_*.csv

A spreadsheet with much the same information as the HTML file.

102_73181_*.bom*

A plain-text rendering of the same information found in the HTML file.

102_73181_*.lbl.*

Printable labels for 14-up Avery labels.

These labels are provided to label bagged parts. The **IDX**(Index) numbers that show up on these labels match up with all of the other files that group parts by value/package.

102_73181_*.mbr.*

Master Build Record.

The *Master Build Record* has one part per detail line. This list may be used to check off parts as they are added to the board. You will note that the **Comp S/N** column has a schematic locator for the part and the **Component MIT** column has the location of the part on the circuit board.

The circuit board location is referenced to the lower left corner of the circuit board as viewed from the top. This means that the X ordinate for the bottom moves from right to left (i.e. backwards).

102_73181_*.DigiKey*.csv

Two dpreadsheets with DigiKey part numbers.

The **BOM Manager** entry will redirect to DigiKey where you can then upload the complete parts list (102_73181_*.DigiKey_bom.csv). The CSV file has three quantity columns to choose from and some quantities will need to be adjusted or substituted. It does save having to enter it all by hand.

4.3 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 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 bias a pair of varactors.

The modulated reference clock is multiplied using an oscillator internal to the clock synthesizer and then using the (SI5351) 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 also available in a UHF variant, allowing the system to be used in the 70cM band.

4.3.1 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 a multiple of the carrier frequency and fed, along with the reference clock, into a DPLL (inside the SI5351). The output of this internal DPLL keeps the internal VCO operating at the target frequency. The VCO frequency is, of course, divided to produce the carrier clock.

The SI5351A built-in oscillator uses an external crystal to generate a stable reference clock to discipline the VCO. 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 may be left unpopulated with the output of the SI5351 being delivered directly to the output filter. Output power using only the digital output of the SI5351 is less than 10mW. This output level should be adequate for short range hunting.

For more power when using the SI5351, look at the 102-73161-28 and 102-73181-28 power amplifiers. These amplifiers may be built with power levels up to a bit over 100mW.

4.3.2 VHF Power Amplifier

The output amplifier is external to the motherboard (one is shown installed in figure 4.1). The mechanical interface is virtually unchanged from the 102-73161-25 board (where the amplifier was first moved off of the motherboard). Pins have been added to support the DRA818/SA818 modules, but earlier power amplifier boards will work with newer motherboards.

Other newer power amplifiers are mechanically compatible with the older boards, but may not perform well with the ICS525 without an input matching network.

At least one of the new power amplifier boards makes use of the split PTT *(push-to-talk)* and PD *(power-down)* connections of the 102-73181-10 board.

4.3.3 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 our fox transmitters the receive section is not used. The DRA818/SA818 module may be pushed into a power-down state or switched off entirely to conserve power between transmissions.

The DRA818/SA818 modules are **not** backwards compatible and may only be used with the 102-73181-10 (or newer) boards. Older revisions are missing discrete control of one of the DRA818/SA818 control pins.

4.3.4 Output Filter

There is an output filter on the main board to reduce harmonics produced by the various amplifiers. The DRA818 and SA818 modules both 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.

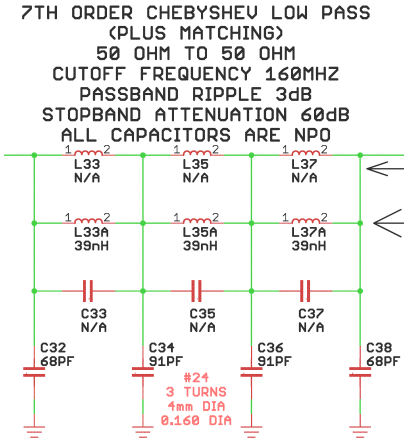


Figure 4.3: SI5351 Output Filter Schematic

Note that the schematic shows duplicate inductors and unpopulated capacitors. This provisioning of extra parts on the circuit board allows the filter configuration and parts selection to change (allowing for alternate filter topology). For example, L33A is a surface mount package and L33 is a thru-hole package.

Other Bands

The following table, found in the 102-73181-10 schematic, contains the calculated filter component values for various cutoff frequencies.

OUTPUT FILTER VALUES (3dB RIPPLE)														
	INPUT	C32	L33	C33	C34	L35	C35	C36	L37	C37	C38	R39	C39	OUTPUT
55 MHZ LPF (7/CHEBYSHEV)	50 Ohm	204PF	112nH		269PF	116nH		269PF	112nH		204PF	DNI	390PF	50 Ohm
110nH = Wurth 744393230011		200	110		270	120		270	110		200			
74 MHZ LPF (7/ELLEPTIC)	50 Ohm	45PF	134nH	7.17PF	63.2PF	87nH	36.5PF	54.5PF	92nH	26.3PF	31.9PF	DNI		50 Ohm
ALL AIAC-1812 (PADS MATCH)		47	120	6.8	68	82	39	47	100	27	33			
146 MHZ LPF (7/CHEBYSHEV)	50 Ohm	76.7PF	42.1nH		101PF	43.8nH		101PF	42.1nH		76.7PF	DNI	390PF	50 Ohm
43nH = Delevan 4426-10		75	43		100	43		100	43		75			
160 MHZ LPF (7/CHEBYSHEV)	50 Ohm	70.0PF	38.4nH		92.3PF	40.0nH		92.3PF	38.4nH		70.0PF	DNI	390PF	50 Ohm
ALL AIAC-1812 (PADS MATCH)		68	39		91	39		91	39		68			
460 MHZ LPF (7/CHEBYSHEV)	50 Ohm	24.4PF	13.4nH		32.1nH	13.9nH		32.1nH	13.4nH		24.4PF			
13nH = Murata LQW2BAN13NG00L		24	13		33	13		33	13		24			

https://rfdesigntools.pythonanywhere.com/tool/filter_design
https://markimicrowave.com/technical-resources/tools/lc-filter-design-tool/

Figure 4.4: Output Filter Table

The table has exact calculated values, shown in black. The tool also calculates and plots using standard values, these being in red. The inductors that are not from the AIAC-1812 series in the parts list, are indicated in the table.

The tool used for the calculation:

`https://markimicrowave.com/technical-resources/tools/lc-filter-design-tool`

160MHz Low Pass

The filter response plot for the 160MHz case from the *Marki Microwave LC Filter Design Tool*:

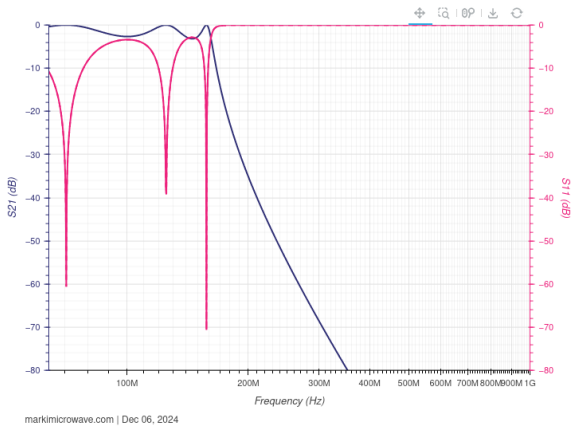


Figure 4.5: 160MHz LPF Filter Response

The filter response plot taken from a NanoVNA-H4:

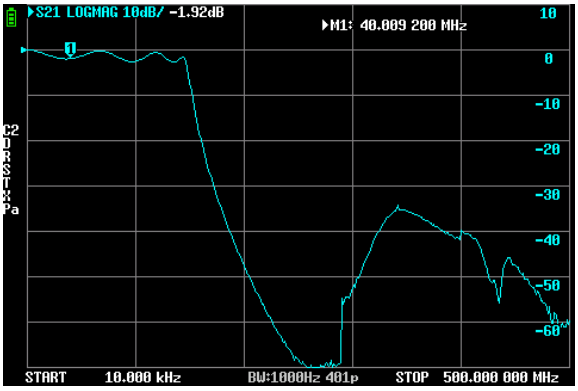


Figure 4.6: VNA Filter Response

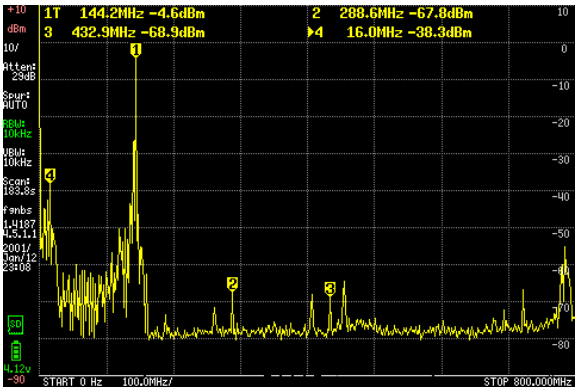


Figure 4.7: TinySA Spectrum

This plot is the fox transmitter sending carrier showing harmonics that make it through the output filter. VBW/RBW is 10KHz.

Comparing figure 4.7 and 4.6 it doesn't look like the Nano-VNA-H4 is properly calibrated above about 280MHz as the TinySA shows good harmonic suppression.

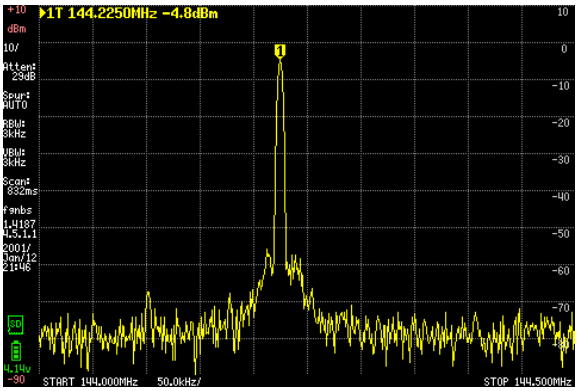


Figure 4.8: TinySA Spectrum 2

Here we are narrowed in on the 2M band. An adjacent H.T. shows carrier present and audio is quiet. VBW/RBW is 3KHz.

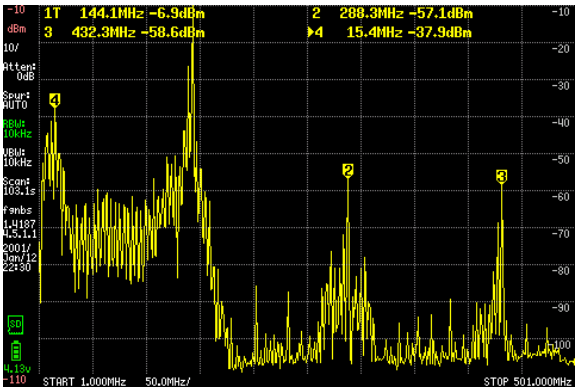


Figure 4.9: TinySA Spectrum 3

Here the scan is taken with higher resolution. VBW/RBW is 10KHz.

74MHz Low Pass

The filter response plot for the 74MHz case from the *Marki Microwave LC Filter Design Tool*:

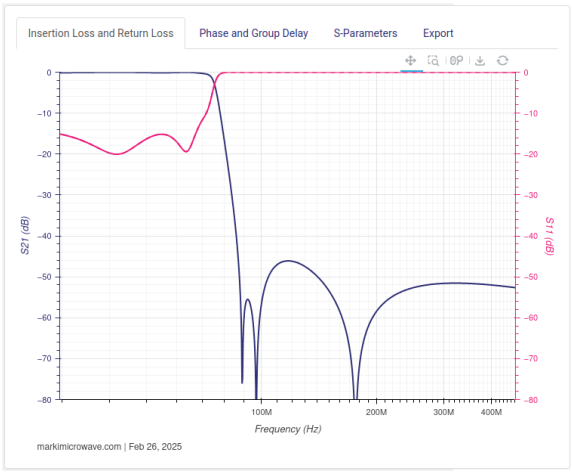


Figure 4.10: 74MHz LPF Filter Response

Using the *Marki Microwave LC Filter Design Tool* a set of values can be found for operating in the 6M band that uses inductors from the *Abracon AIAC-1812* series of inductors.

This has not been tested by the author.

4.4 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. The table for the ICS525, covers a limited frequency range sue to limitations of the ICS525. Thes table for the SI5351 covers a very limited frequency range, sufficient only for characterizing the frequency offset of the reference oscillator.

The software supports a method of saving register values in external memory (i.e. FRAM). Operating on frequencies outside those provided in the table is accomplished by storing register values for additional frequencies in the external memory. The external memory can be loaded through the control port (this does not require a zNEO programmer).
The external table may be used to produce a carrier in the 6M band, if desired.

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

4.4.1 SI5351

The SI5351 requires us to come up with a set of control patterns to be loaded into the (somewhat less than 188) device registers.

Skyworks provides a clock configuration tool for the SI5351. We also have a utility to do the same work, generating zNEO source code and Fox Transmitter setup commands.

The plan is to keep a small subset of frequencies in the zNEO program flash and provide any additional register patterns using the command interface.

> With the 3.73 and later software, the table has 144.100MHz, 144.150MHz, 144.200MHz, and 144.300MHz. This should provide a few spots that can be used to measure the crystal offset.
>
> With the 3.80 and later software, the table was spread across the band with zero-offset points at 144.100MHz, 144.500MHz, 145.000MHz, 145.500MHz, 146.000MHz, 146.500MHz, 147.000MHz and 147.500MHz. This set of points should allow characterization of the crystal offset across the 2M band.

This approach demands that an external frequency table be loaded for proper operation. Although you are free to generate register patterns for any arbitrary frequency, your hunters will be using commercially available radios with 5KHz channel spacing.

> The FRAM on *FOX21* through *FOX32* is loaded with an external frequency table, in 5KHz steps, to cover between 144.100MHz and 144.345MHZ. This *external frequency table* accounts for the error in the reference oscillator.

Keep in mind that there are no interlocks in the software to limit the frequency we select! It is the operators responsibility to configure the SI5351 to operate within the frequency allocations of the operators license!

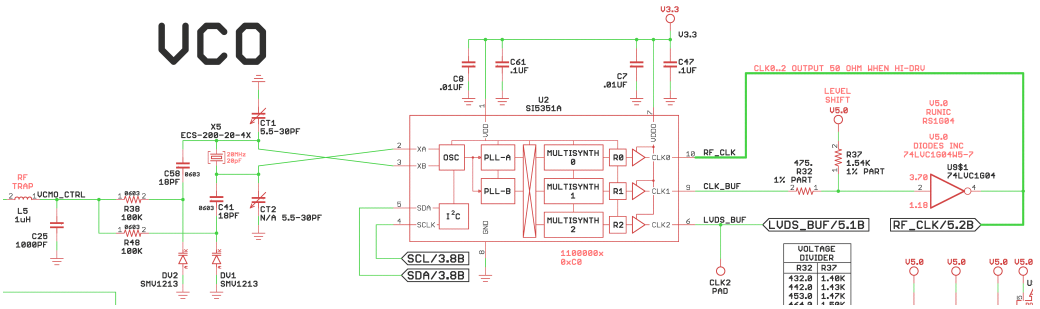


Figure 4.11: SI5351 Schematic

On the left you will find the FM modulation scheme introduced on the 102-73161-25 board. A pair of varactor 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 remaining synthesizers are configured in their power-down state to reduce power consumption.

Also take note that we deviate from the nominal crystal value for the SI5351 as suggested in the datasheet. For our design, we choose to make use of the same crystal for the zNEO and the SI5351. The *Multisynth* register values must then be calculated based on the 20MHz crystal.

The clock output that is in use is enabled using a command sent through the I^2C 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 16.

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 the RF pin on the daughter board connector. If U4/U9 are populated, a trace cut is required on the bottom side of the circuit board.

4.4.2 DRA818/SA818

The DRA818/SA818 modules differ in that they are self contained radio subsystems. Frequency selection is handled in the module, not by the clock synthesizer on the motherboard. The motherboard synthesizer does not need to be populated when using this RF module. Note that a 1½W or 2 W 5V regulator is required when using these modules.

The module is controlled using the 2nd. UART channel (it is no longer routed to the time network). The DRA818/SA818 module control scheme is fully implements on the 102-73181-10 boards only. Earlier board revisions do not implement the required power and control elements for the DRA818/SA818 module to be correctly controlled.

As mentioned elsewhere, the time network function is not implemented in this version of the software.

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

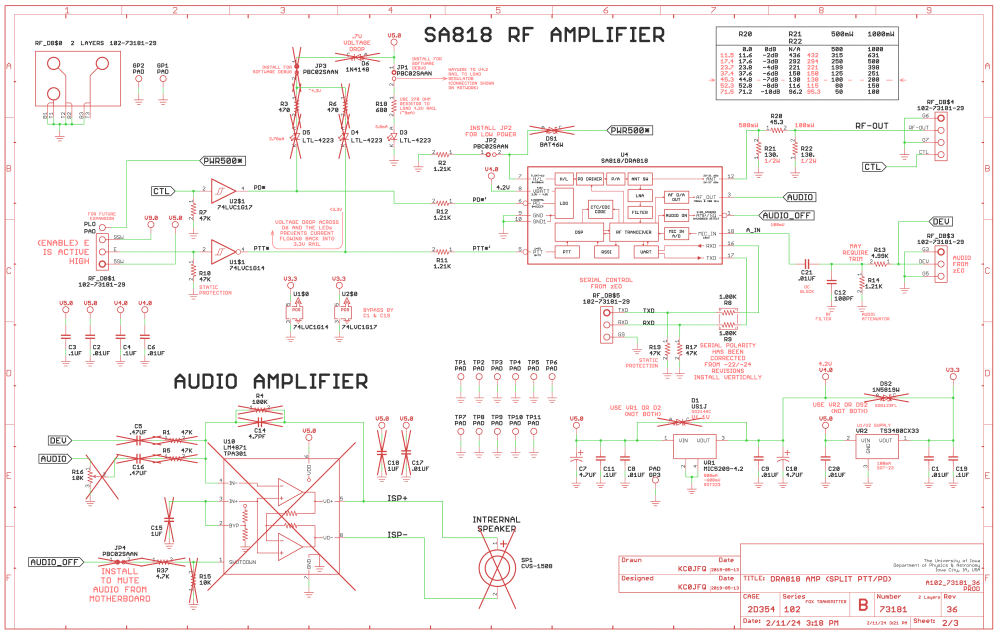


Figure 4.12: DRA818 Daughter board Schematic

This is the marked up schematic. This is how a production daughterboard is built. This reflects how the board in figure 4.13 on page 40 is built.

The audio amplifier and indicator lights are not needed so are not installed (R18 and D3, although shown on the schematic, are not installed).

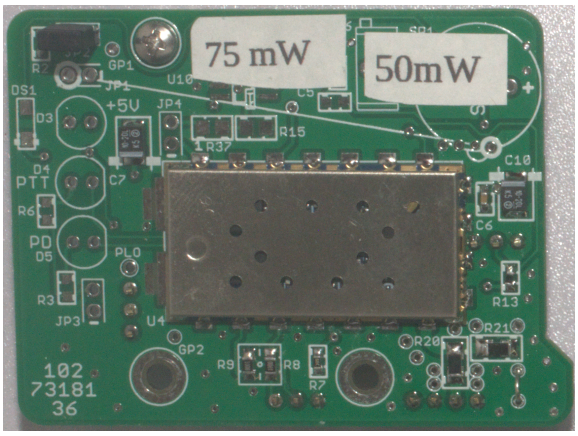


Figure 4.13: DRA818 Daughter board

This is a functional DRA818 board. This is the latest revision.

The 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 correctly 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 independently. The V3.27 and later software correctly supports power sequencing on the 102-73181-10 board.

Note that the SA818S module is specified to operate at 5.0V whereas the DRA818 module operates at 4.2V. The SA818S module therefor has the potential to produce slightly more output power.

When using the SA818S module VR1 can be eliminated. D1 may then be replace with an inductor or a simple jumper wire.

4.4.3 ICS525

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

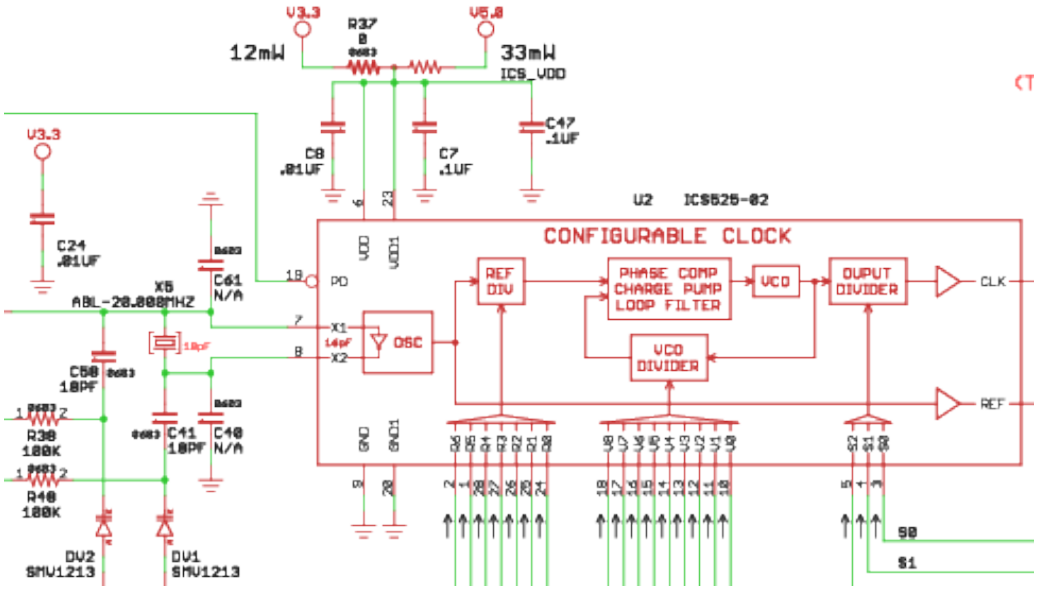


Figure 4.14: ICS525 Schematic

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 4.11 on page 38.

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

4.4.4 ICS307

Not currently supported as no 102-73181-0 boards were built (these use the ICS307 device(.

4.5 Transmit Timing

A transmission cycle is initiated when a **BEGN** command (table 10.32 on page 193) is executed. This command turns on the RF system and waits for it to stabilize. Once operating steady-state and station identification message (i.e. the station callsign) is transmitted.

Transmission is terminated after a **DONE** command (table 10.37 on page 198) is executed. This command causes a station identification message to be sent before the RF system is shutdown.

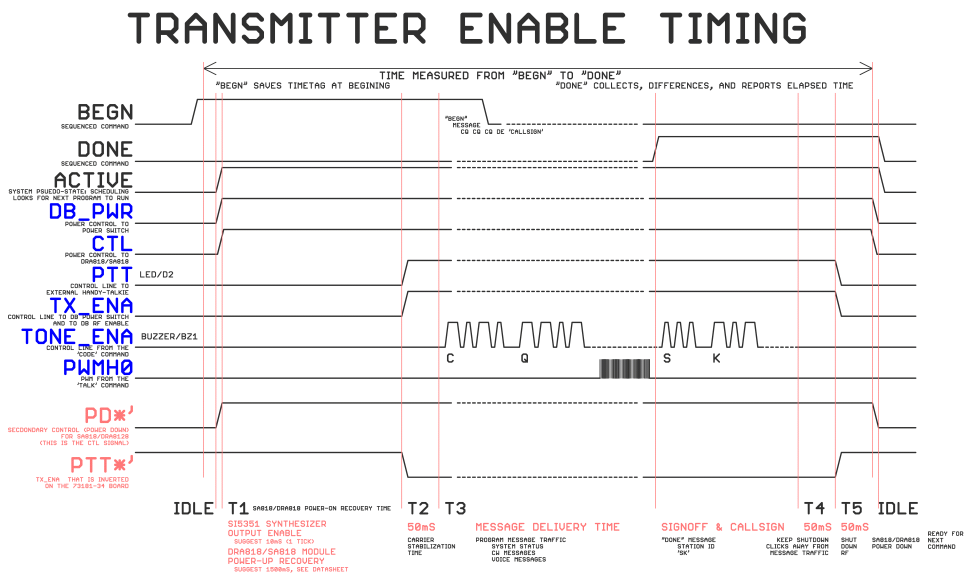


Figure 4.15: Transmit Timing

Assuming the document has been printed in color, the **black** labels indicate commands (in the FRAM), the **blue** labels indicate signals on the main board and **red** labels indicate signals on the RF daughterboard.

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 CQ 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:

```
de <call> 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/DRA818 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 DRA818 daughter board.

PTT*'

DRA818 Signal Net

This is the **TX_ENA** motherboard signal, which is positive true, with its logic sense switched to match the requirement of the DRA818.

T0 IDLE

System Idle.

The fox transmitter system in in the T0 state when waiting for a scheduling point to occur.

Most of this time is spent with the zNEO executing a **HALT** instruction to minimize power. Commands will be proceesed in the T0 state. Interrupts continue to be proceesed so the system time updates.

T1

DRA818 module wakeup time.

This is the time needed for the DRA818 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.

4.6 SI5351 Synthesis

The SI5351 synthesizer was added to the 102-73181-5 update and provides a more generic (flexible) means of synthesizing a carrier. Unlike the ICS525 and ICS307, we aren't locked into just a few useful frequemcies.

Additional discussion concerning configuring the SI5351 may be found in section 17 on page 295. The software in the Fox Transmitter does not perform the calculations necessary to generate the register patterns, rather we rely on a simple table lookup. This lookup scans the external FRAM and the internal program flash. This discussion covers the steps used by the external utility that generates the frequency setup tables.

4.6.1 SI5351 Block Diagram

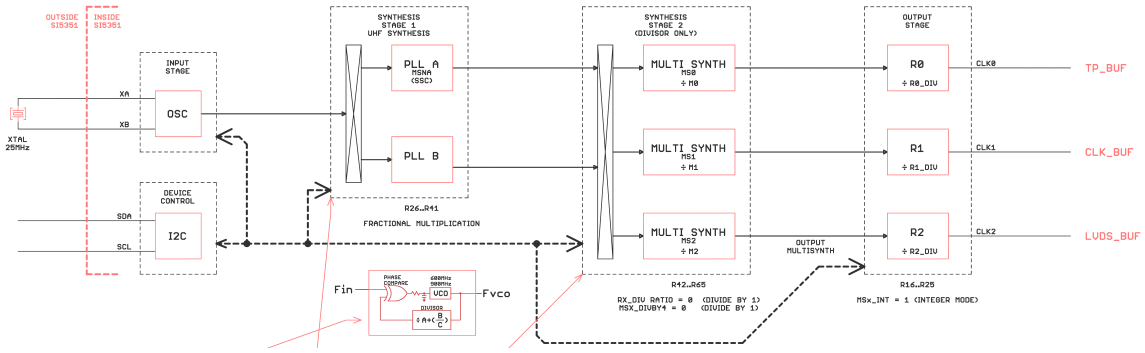


Figure 4.16: SI5351 Synthesizer Block Diagram

Working from the left, we have the reference oscillator (**Input Stage**) that generates the (externally modulated) 20MHz (not 25MHz or 27MHz as described in the datasheet) reference clock (see section 4.11 on page 38). This reference clock is passed on to *Synthesis Stage 1*

Synthesis Stage 1 consists of the DPLL, VCO and *Multisynth* that are MSNA. The VCO operates up in the the UHF range (roughly 600MHz to 900MHz). MSNA is configured to operate the VCO at an integer multiple of the carrier.
The DPLL divisor in MSNA (which need not be an integer) is configured to divide the VCO clock (which is operating at an integer multiple of the target carrier frequency) output down to match the 20MHz reference.
MSNB (PLL B) is not used at all, so it is left powered down.

Although operating MSNA with a fractional divisor introduces some phase noise, this is not concerning for operating FM. We introduce considerable noise (i.e. audio modulation) at the reference crystal.

Synthesis Stage 2 are the MS0, MS1, and MS2 synthesizers. These all configured to divide the MSNA output (an integer multiple of the carrier) down to the carrier frequency.
By keeping MS0, MS1, and MS2 set to integer divisors, we avoid introducing additional phase noise at this stage.

The **Output Stage** are the output dividers R0, R1, and R2.
These are set to divide by one, making CLK0, CLK1, and CLK2 all at the carrier frequency.

Only one of the three Stage-2 and Output-Stage paths are enabled. The unused channels remain in a power-down state to conserve power.

Consult the source code to the table generation utility in section 17 on page 295 for specific limits on the operation of the VCO.

Shown below the *Synthesis Stage 1* block is a (very) rough approximation of the PLL, VCO and *Multisynth* divider. This block incorporates the first fractional divider (called **MSNA/PLL-A** and **MSNB/PLL-B**) required to divide the VCO frequency down to match the reference crystal frequency.

In our Fox Transmitter, we make use only of **MSNA/PLL-A**, leaving **MSNB/PLL-B** in a power down state to save power.

In *Synthesis Stage 2*, we see three more instantiations of the *Multisynth* divider block used to divide the VFO output down to the target carrier frequency (or a nice integer multiple thereof). Keep in mind that the input to this stage is the internal VCO, so the frequency range of the input is the roughly 600MHz to 900MHz mentioned in the previous paragraph.

In the *Output Stage*, we may divide by an integer divisor, but for the Fox Transmitter we fix this stage as a divide by 1.

We also select the appropriate output buffer configuration. As suggested in the drawings, only one of the three outputs may be enabled for correct operation. The unused *Multisynth* blocks are left powered down as you would expect to save power.

4.6.2 SI5351 VCO Frequency

First we must come up with a frequency plan, targeting a good spot to operate the VCO.

For the Fox Transmitter, we take a direct approach by attempting to choose a VCO frequency that is an even multiple of the carrier.

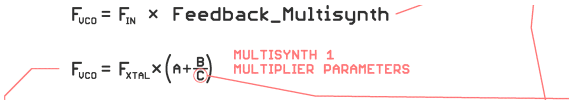


Figure 4.17: SI5351 Synthesizer VCO Selection

4.6.3 SI5351 MSNA fraction

The MSNA divisor, then, will be calculated from the VCO frequency in order to divide the VCO frequency down to match the reference crystal frequency.

We use this calculation to come up with the values for the three *Multisynth* registers.

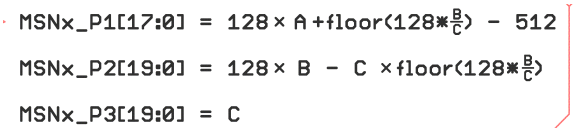


Figure 4.18: SI5351 Synthesizer Register Calculation D

4.6.4 SI5351 M0/M1/M2 fraction calculation

The divisor for the M0/M1/M2 *Multisynth*, are all the same and are usually easy to calculate as this is a simple integer division.

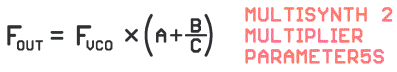


Figure 4.19: SI5351 Synthesizer Register Calculation E

As we have chosen to keep an integer divisor here, the values for the MSx_* registers are simple to come up with. The value for MSx_P2 is always zero and the value for MSx_P3 is always one for the 2M band. The value for MSx_P1 will generzally come out to a value of 256 for the 2M band.

$$MSX\_P1[17:0] = 128 \times A + floor\left(128 * \frac{B}{C}\right) - 512$$

$$MSx\_P2[19:0] = 128 \times B - C \times floor\left(128 * \frac{B}{C}\right)$$

$$MSx\_P3[19:0] = C$$

Figure 4.20: SI5351 Synthesizer Register Calculation F

4.6.5 SI5351 R0/R1/R2 divisors

We target a divisor of one or two here.
When operating in the 2M band, we simply fix the divisor with a value of one.
The divisor can be set to an even number to operate down in the HF bands (specifically 50M).

4.7 Voice

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

After some *deep thought* following a question from W0PPF the answer began to percolate to the surface (*42*, the answer to life the universe and everything). 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 around 2KHz to 2.5KHz of bandwidth.
The zNEO is extremely SRAM limited, so the waveform data is not buffered in the zNEO but is processed a single sample at a time. The sample rate is controlled by the SPI shift clock, this clock being configured to operate at 32KHz for the 4KHz rate (or 40KHz for q 5KHz rate) resulting in a new sample from the FLASH device becoming available every 200uS to 250uS.

A source 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 slower sample rates (4KHz to 8KHz) result in understandable speech. The slower sample rates also allows a reasonable amount of voice storage.

Moving to a 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 the audio 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 used in other applications 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 thereby increasing load times.

4.7.1 Audio File System

The *Audio File System* is split into directory records that reside in the FRAM device, and waveform data which is held in the FLASH device

> This seperation comes about due to the way that FRAM and FLASH are managed. The FRAM driver deals with 32 byte records and does not have to account for device erase time constraints. The FLASH driver deals only with InTel HEX records. Furthermore, the search algorithm looks only in the FRAM for records as a search of FLASH memory could take quite some time due to its size.
>
> 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 Data* is stored in the FLASH device as a WAV file (i.e. binary data). Each file consisting of some multiple of 32 bytes as 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 expected to deal with the upper bits of the address.

Nominally the *InTel HEX* record is 32 bytes of image data. Shorter data records are processed by the loader but the 32 byte length is chosen to minimize download times. Buffer size constraints dictate the 32 byte record length, longer records will overflow the input buffer.

The command decoder recognizes an *InTel HEX* record by its leading colon (:) character. The decoder also discards embedded space characters allowing it to process the expanded hex file line shown here as well as a line formatted in a standard manner.

Example portions of an audio file:

```
:02 0000 04 0000 FA
:20 0000 00 52494646501000005741564566D74201000000001000100A00F0000A00F0000 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 data (i.e. the InTel HEX records) and a directory entry describing location of the audio file.

This example fragment is part of the vocalization of the W0JV callsign. This audio fragment is RIFF/WAVE formatted, so the length and sample rate are in the RIFF/WAVE header. You should be able to pick out "RIFF" (0x52 0x49 0x46 0x46) at the begining.

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 4.1: 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 audio utility inserts white-space into the InTel HEX records to delineate the columns indicated in Table 4.1 to improve readability.

Table 4.2: 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 **TALK=** 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 passing it to 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.

4.7.2 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 FLASH and the directory records loaded 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 the *InTel Hex* records containing the waveform data, and the directory records that contains the starting address of the waveform file in FLASH.

The raw input files are produced using *sox* to re-sample the audio clip, typically from 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.

4.8 TOY Clock

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

Inadvertant power loss in the field does not affect scheduling, simply restore power and the fox transmitter will send message traffic on schedule.

4.8.1 The DS1672

The DS1672 is a simple TOY (Time-of-Year) clock. It has a 32KHz oscillator (requiring an external crystal) and divider. A 32 bit seconds counter is accessed over the I^2C interface. This 32 bit register may be loaded by the zNEO to set the system time. No conversions need be performed as the DS1672 time register format is identical to the system time format (truncated UNIX time).

The DS1672 allows for 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.

4.8.2 Reading from the DS1672

Some units seem to exhibit a problem when first reading the DS1672. This shows up as reading all zeros from the time register leaving the fox transmitter system without a useful time.

This, of course, prevents the fox from operating on its assigned schedule. It runs, but with a bad time that isn't in sync with the rest of the transmitters.

To address this issue, the DS1672 must read twice with a small delay between reads. A delay of 1/2 second between reads seems to be adequate.

Initialization Example:

```
esav INI=TIME
esav INI=WAIT 0.5
esav INI=TIME
esav INI=EPOC -5.0
```

Table 4.3: DS1672 Register Map

Addr	B7	B6	B5	B4	B3	B2	B1	B0	Function
+00	D7	D6	D5	D4	D3	D2	D1	D0	Counter Byte 0
+01	D15	D14	D13	D12	D11	D10	D9	D8	Counter Byte 1
+02	D23	D22	D21	D20	D19	D18	D17	D16	Counter Byte 2
+03	D31	D30	D29	D28	D27	D26	D25	D24	Counter Byte 3
+04	EOSC								Control
+05	TCS	TCS	TCS	TCS	DS	DS	RS	RS	Charge

Four registers (0..3) hold the current seconds count.

The fifth register (4) is the oscillator enable bit; zero to enable the oscillator.

The sixth register (5) are the charge control bits; they may all be set to zero to disable the DS1672 charge function.

4.8.3 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. Given a typical AAA cell with a capacity of 1000mAH, the AAA battery will last over 5 years powering the DS1672.

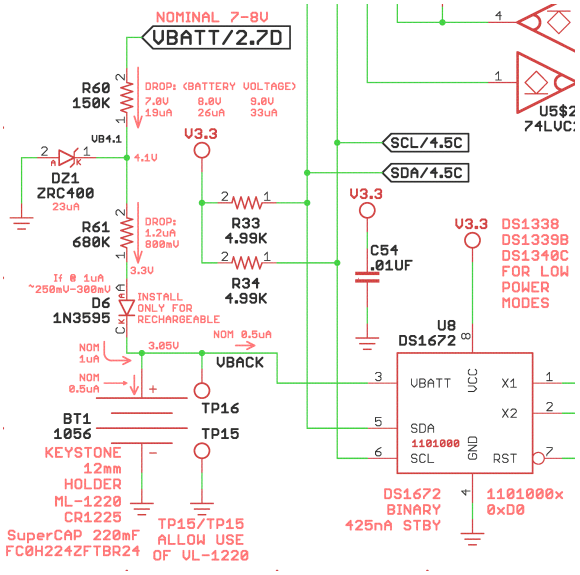


Figure 4.21: DS1672 Charge Control

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 to the DS1672 when the zNEO will not run, the fox transmitter should not be left with discharged batteries in order to avoid battery leaks that will cause damage to the circuit board.

DZ1 is a simple 4V shunt reference. DZ1 keeps its cathode near 4.1 volts when supplied with some minimum amount of current.

DZ1 Selection

The chosen reference device, the ZRC400, comes in an SOT23 package requiring only 18µA to 23 µA to stay in regulation.

One pin compatible device is the LM4040 which requires about 70 micro-amps through the device to maintain regulation. Changing to this device would require a change to R60 (to about 56K) to supply additional current to keep the DZ1 cathode at the target 4.1V.

As battery voltage drops below 7 volts, the ZRC400 cathode will start to fall below the specified minimum of 4.1 volts. This, in turn, reduces the currrent that can be supplied across R61/D6 which will then allow current from BT1 to flow into U8. When BT1 is installed as a coin cell holder and we are using a CR1220 cell (37mAH) and assuming the DS1672 current draw is less than 500nA, we have a backup time of 8 years.

Using SuperCAP for BT1

You may substitute a *Super-CAP* in place of the TOY clock backup battery. This device, **FC0H224ZFTBR24**, will mechanically fit on the battery pads. Some Kapton tape should be placed under the cap to isolate traces and vias that could make contact with pads on the *Super-CAP*.

As the available current to the backup battery is very low, on the order of a micro-amp, it will take considerable time to charge the cap (on the order of a month!).

We can improve that somewhat by reducing the capacitors electrical size (a mechanically smaller may not reach the mounting pads). A 100mF device **FC0H104ZFTBR24** and a 47mF device **FC0H473ZFTBR24** are listed at DigiKey.

These values are **millifarads** not microfarads!

The expected run time here would be similar to the charge time as we expect the discharge current to be a bit less than half of the available charge current. This should give us days of operation.

Using Aluminum Electrolytic for BT1

We may also substitute a plain old aluminum electrolytic place of the TOY clock backup battery assuming we keep the main battery installed from time we set the clock through to the end of the hunt. A typical aluminum electrolytic device, such as **EEE-FK1A102P**, is mechanically similar to the Super-CAP above. Some Kapton tape will be required here.

A larger device, **EEE-FK1A222AQ**, may also be employed. This capacitor is mechanically a bit of a better fit to the pads on the circuit board. Some Kapton tape is required here as well.

The limited current in this scenario presents much less of a charging issue as we as dealing with 1mF capacitor. The charge time in these cases is nominally less than 1 hour after the main battery is connected.

The expected run time here would be similar to the charge time as we expect the discharge current to be a bit less than half of the available charge current.

In this case, assume we keep the main battery in the transmitter after the time has been set (not at all unreasonable). If we need to change the battery in the field, the 1000uF capacitor would provide tens of minutes of backup time to changeout the battery.

BT1 not installed

We can probably get away without installing a backup battery at all. The external charge control circuit (R60, DZ1, R61, D6) can be modified by substituting a **ZRC330F01TA** shunt reference (3.3V) for DZ1 (which was a 4.1V) and lowering the resistance or R61 to around 47K. This substitution allows the regulator circuit to provide 3 volts to the DS1672 through the main battery.

A long as the main battery is well connected, the TOY clock should function and track time.

Keep in mind that noise on the main battery net (such as from a loose connection) has the potential to glitch the TOY clock and take out our time reference.

4.8.4 System Time

The system time, maintained by the zNEO, is kept in the same format as that stored in the DS1672. No conversion is required when retrieving the time data. Neither is any conversion requires when storing time into the DS1672.

4.9 Deviation Control

Carrier deviation signal generation.

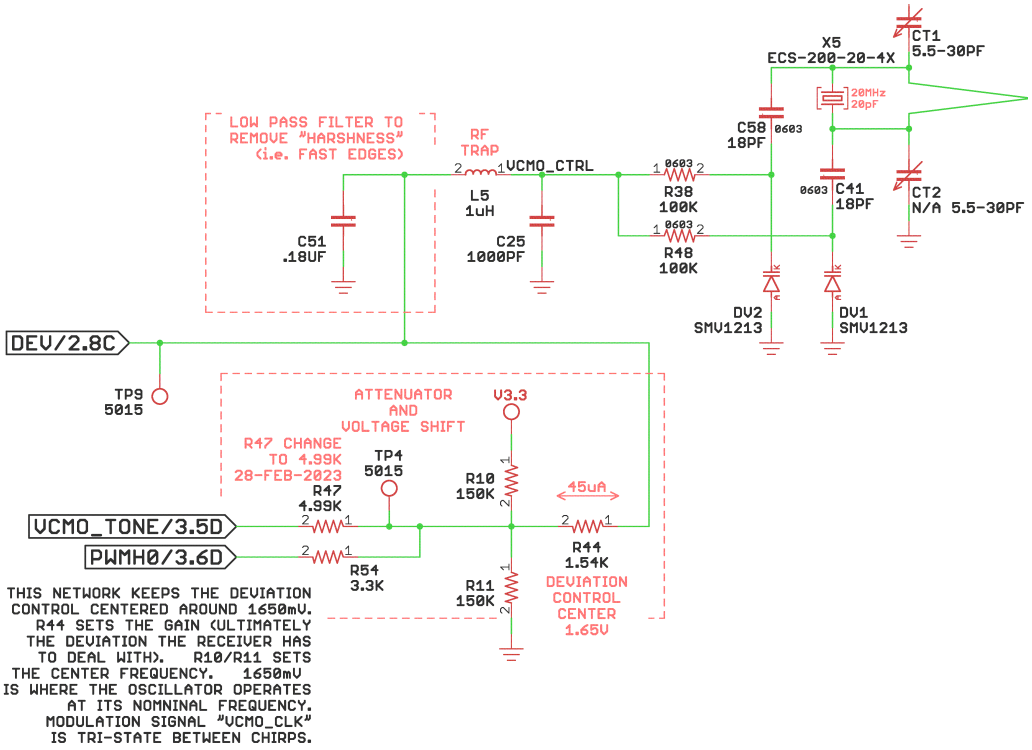


Figure 4.22: Deviation

The output from R38/R48 connects to the crystal load capacitors shown in figure 4.11 on page 38.

VCMO_TONE is the CW signal from the square wave generator (in the zNEO). The gain through this path is controlled by R47 (R10/R11 impedance is 75K).

PWMH0 is the audio signal from the PWM channel in the zNEO. The gain through this path is controlled by R54.

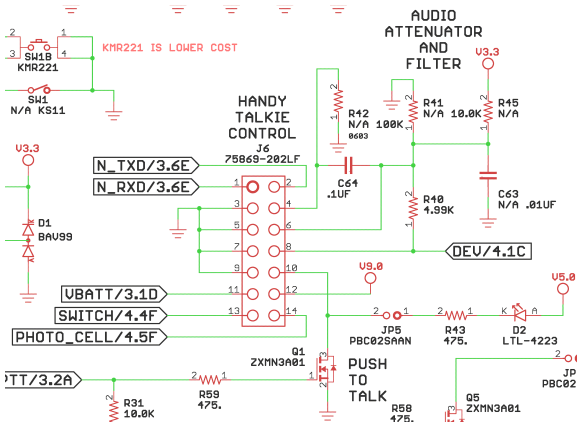


Figure 4.23: External Connection

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 daughter-board.

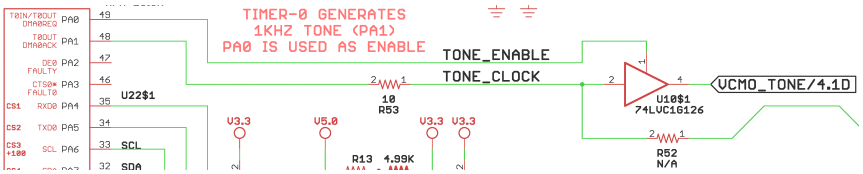


Figure 4.24: VCMO_TONE

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). The **TONE_ENABLE** net also runs to the buzzer that may be used for debugging. send code.

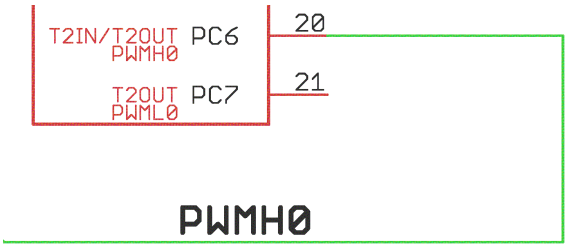


Figure 4.25: PWMH0

The audio signal is produced by the PWM controller in the zNEO. PC6 is configured as the output of the PWM block. Once the PWM controller is configured, waveform data is read one eight bit sample at-a-time from the FLASH device and loaded into the PWM control register. As mentioned elsewhere, sample rate is controlled by the SPI clock.

4.10 Power

The device is aimed at being able to run on a LR22 9V alkaline battery for the duration of a hunt assuming a 20% duty cycle. The limited current delivery capacity of the LR22 limits the achievable output power, so choose the PA to limit current consumption (consider running *barefoot* using the 102-73161-22 *Amp Bypass* board).

The circuit board, when installed in the case specified in the schematic, allows for using a AAA pack to increase battery life to about 24 hours with a 100mW amplifier. 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 used 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 low cost 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 DRA818/SA818 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 DRA818/SA818 RF board, much less being able to transmit. Using the DRA818/SA818 also drives the regulator selection; it must be capable of delivering enough current for proper operation.

4.10.1 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 conversion 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 10 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.

BATR I

Vertsion 3.91 and later.

This command was added to the *BATtery Report* command to dump the coefficients table. A recalculation in December of 2024 changed some target values and recalculated the coefficients table.

4.10.2 Power Plot

The addition of the **BATR** command allows for the collection of battery data when the transmitter is in operation. This example is a typical transmitter with a new battery and a power amplifier daughterboard that produces about 60mW. The cycle is 360 seconds (6 minutes) with a 55 second active time.

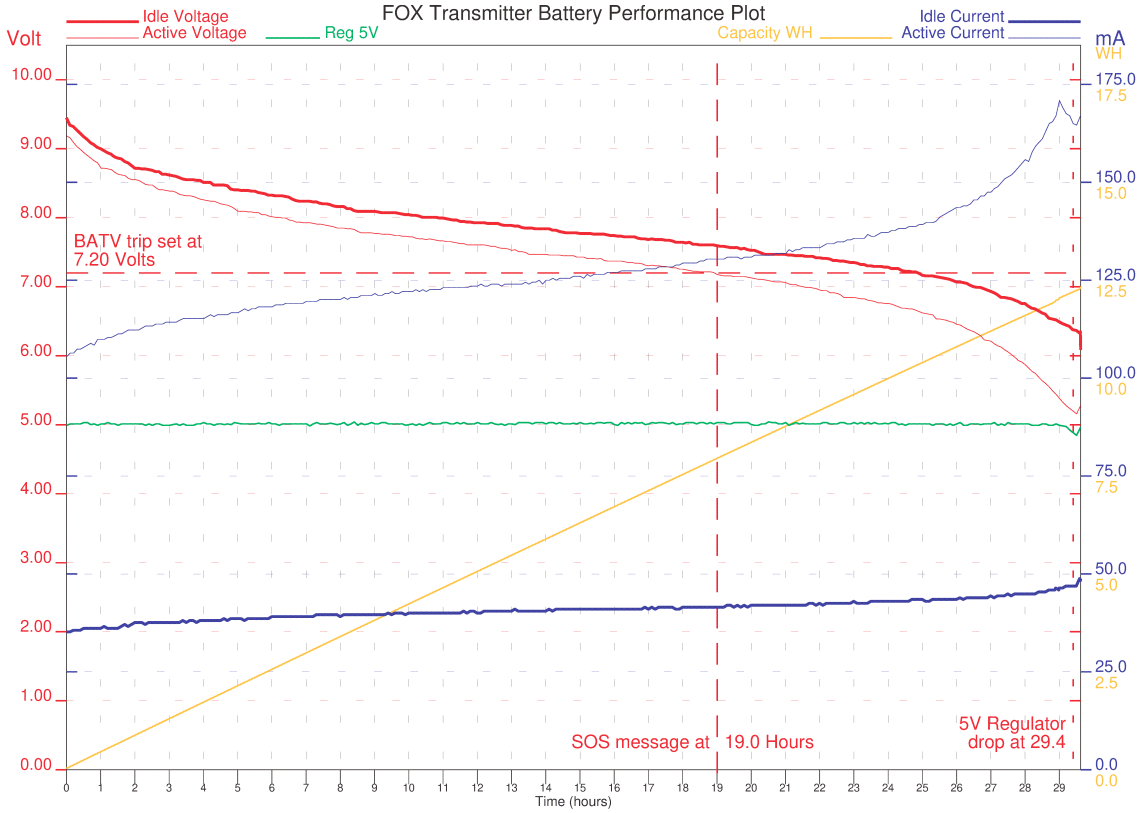


Figure 4.26: Power Plot

A **BATR** command is placed at the begining of the (**S0=**) sequence, before the power amplifier is powered (i.e. before the **BEGN** command). A second **BATR** command is placed somewhere in the active message (between the **BEGN** command and the **DONE** command) to catch the current draw when transmitting (i.e. the power amplifier is powered and actively transmitting RF).

The *Fox Transmitter* is then connected to a dummy load and to a host computer to log the activity (that is coming out on the command port) and left on until the battery collapses.

When the battery voltage, during an active state, drops below what the regulator will handle, expect the data collection process to falter.

The VX7805-500 data sheet indicates a minimum input voltage of 6.5V with the higher current devices indicating the need for a higher minimum input voltage.

The battery report appears something like the following:

```
sts47,00* Handler_BATR (cmd_battery.c*) V=7.840[0324] I=115.7[00ED] 5=5.012[0202] State-T3 0.01 Sec
```

The handler name uniquely identifies the battery state report (i.e. the **Handler_BATR** is unique to the **BATR** command) and provides the voltage, current, and transmitter state.

The transmitter state will be either **State-T0** to indicate and idle system or **State-T3** to indicate the system is transmitting.

The logging program is expected to provide a timetag as the data is collected (as is the case with the *halo_term* utility used by the author) when running software versions prior to V3.90. V3.90 update adds a timetag (in decimal seconds) to the report eliminating the need for the logging software to deal with this.

We monitor the 5 volt regulator channel to see when the regulator falls out of regulation. The **T3** state reading is plotted, this being when the 5 volt regulator is under load (near dropout).

The zNEO runs off of the 3.3V rail, so it will continue to operate until the 3.3V regulator drops out (the datasheet suggests that this is around 4.5V to 4.8V). We can then look for the regulated 5V line to drop to determine the end voltage point and work back towards the warning voltage to use in the battery reporting commands.

4.10.3 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** surface mount part on the bottom side 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.

4.10.4 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 (i.e. to the left).

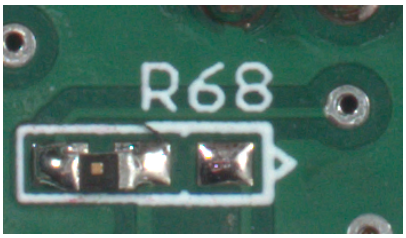


Figure 4.27: R68

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).

4.11 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 4.28: TTL-232R-3V3-AJ

Figure 4.29: 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.

4.12 Configuration Order

Configuration commands are somewhat order sensitive. A list of available **CONF** commands may be obtained by entering the **CONF** command with no arguments. The commands are detailed in section 10.2.8 on page 172.

The *callsign* and *nickname* should be located near the beginning of the **INI=** file. This establishes the identity of the unit for all commands that require parameter substitutions that follow.

Following RF hardware selection (for example: **CONF SI5351**), the configuration bits are preset to appropriate values for the selected RF subsystem. Additional **CONF** commands may be issued to alter the RF configuration bit settings when required. The commands to do this are in the *SYNTH* group.

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

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

4.13 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 4.4: 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.

4.14 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 this part number is the package type, allowing the software to easily determine when we are running on an older hardware revision (80 pin package).

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.

4.15 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 (at least as this document is being produced).

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. The zNEO has an architecture that is better tailored for use with compilers. The zNEO chip is available with 128KB flash and 4KB SRAM. Although smaller memory footprint devices are available, the current software is a tight fit for the 129KB footprint.

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

4.15.1 Program Structure

The operating software consists of multiple foreground interrupt routines that handle *periodic activities* and a background loop that processes commands and the *background schedule*. The interrupt routines handle incoming serial traffic (i.e. foreground commands), regular interrupts (i.e. the 10mS RTI tick that tracks time) and the periodic interrupt for sending CW traffic.

Command Loop

This is the control loop that controls the system.
The control loop looks for:

1. Command traffic from the control port (i.e. the 3.5mm serial port).
2. A scheduling point (i.e. it is now time to run a sequence).

At the bottom of the control loop the processor is halted to reduce power consumption. The processor resumes execution following the next interrupt (from any source) with control returning to the top of this simple loop.

Command Dispatch

Command dispatch takes the 4 character command stem and performs a table lookup searching for a matching stem. If the match is successful the associated command handler is called to decode and execute the command.

Listing 4.1: Main Loop Command

```
while(1){                                                          823
    //                                                             824
    // clear UART_RX0_Flag before processing commands              825
    // because we're about to clear the read buffer                826
    //    ANY traffic that comes in will abort whatever            827
    //  is active and return control to this main loop...          828
    UART_RX0_Flag = 0;                                             829
    //                                                             830
    j = process_commands();                                        831
    rti=TIMER_get_time(&seconds);                                  832
    //                                                             833
```

Start of the main loop.

Fetch complete line from serial interrupt handler (line 831).

Log the current time so we can document command execution time (line 832).

Listing 4.2: Main Loop Schedule

```
        //                                                                    864
        //  Examine the schedule and run program is it's time!                865
        //    "schedule name" = FOX_Schedule();                               866
        //    returns the schedule name, all ready for                        867
        //    the command processor                                           868
        //                                                                    869
        if((ctemp=FOX_Schedule_Loop(fox_config.RUN_Start))){                  870
            if(fox_config.Configuration_Flags & FOX_CONFIG_DEBUG_SCHED){      871
                sprintf((char*)uart_temp_buffer, R"DBG %s/%d ", Format_Main   872
                    ↪ [0].mat, __LINE__);
                UART_write_USB(uart_temp_buffer);                             873
                sprintf((char*)uart_temp_buffer, R" %ld.%02d0 ",seconds, rti) 874
                    ↪ ;
                UART_write_USB(uart_temp_buffer);                             875
                sprintf((char*)uart_temp_buffer, R"RUN %s", ctemp);           876
                UART_write_USB_CRLF(uart_temp_buffer);                        877
            }                                                                 878
            COMMAND_command_load(ctemp, 0);                                   879
        }                                                                     880
```

Line 870 tests for an active scheduling point. If there is an active schedule that needs attention, the command parser is called with the name of the schedule to run the commands in the specified schedule file.

Listing 4.3: Main Loop Halt

```
        //                                                                    902
        //    ****    ****    ****    ****    ****    ****    ****    ****    903
        // Nothing Happening...                                               904
        //                                                                    905
        asm("\tHALT");                                                        906
    } // while(1){                                                           907
```

At the end of the loop, we run the zNEO **HALT** instruction to stop most activity in the zNEO until the next interrupt occurs.

Command Dispatch

This is a short extract from the command dispatch table

Listing 4.4: Command Dispatch Table

```
                                                NULL};

//
//    COMMAND Dispatch Table:
//      connect command mnemonic to the routine that handles it...
//      Count from 1 (i.e. entry ZERO is a NULL entry)
//      Command Index is the position in this table
//        this is also the status returned when the command executes
//      The command buffer, in its entirity,
//
//struct COMMANDS3 {
//                  BYTE   command_type;
//                  char  *command_mnemonic;
//                  char  *command_help_args;
//                  char  *command_help_text;
//                  int  (*command_execute)(int sts, char *input_buf, char *
    ↪ status_buf, int *arg1);
//                  };
///
 rom struct COMMANDS3 command_table[] = {
//      command-type       mnemonic  args    help text
    ↪                              handler address
//                                   ** limit string length !!! ***
        {COMMAND_TYPE_NULL, NULL,      NULL,    NULL,
            ↪                              &Handler_NULL},          //
//
//-------------------------------------------------------------------------------------------------------------------------------
    ↪
//
        {COMMAND_TYPE_SYS,     R"HELP",   NULL,
                                                R"Help Menu and Items",
                                                    ↪            &Handler_HELP},
                                                    ↪          //
        {COMMAND_TYPE_SYS,     R"HELP",   R"<string>",
                                                R"matching help items",
                                                    ↪            &Handler_HELP},
                                                    ↪          //
        {COMMAND_TYPE_SYS,     R"ONCE",   R"<name>",
```

The comments in the code describe the table organization.
First command, with an index of zero, is used as a *not found* condition.
The help text is buried in this table.

Listing 4.5: Command Dispatch Table Search

```
    //                                                                    398
    return delta;                                                         399
}                                                                         400
int COMMAND_command_index(char *mnemonic){                                401
    int index;                                                            402
    index = 1;  // 0th. element is NULL !!!!                              403
    while(command_table[index].command_mnemonic) {                        404
        if(!strncmp(command_table[index].command_mnemonic, mnemonic, 4)) {  405
            return index;                                                 406
        }                                                                 407
        index++;                                                          408
```

Scanning the dispatch table for matching command mnemonic.

Line 400 starts the scan **after** the 0th. (NULL) element.
Line 401 test for end of table.
Line 402 compares the provided command with the table entry.

Listing 4.6: Command Dispatch InTel HEX

```
    //                                                                    454
    //                                                                    455
    //                                                                    456
    if(ibuf[0] == ':'){                                                   457
        sprintf(Stat, R"sts%02d,%02d* %s ",                               458
                COMMAND_TYPE_FLASH_HEX,                                   459
```

Special case for a leading colon. The colon is expected to indicate a HEX record.
Line 457 is the call to decode and save the contents of the HEX record.

Listing 4.7: Command Dispatch Execute

```
                    UART_write_USB_CRLF(uart_temp_buffer);                582
                }                                                         583
#endif                                                                    584
                //                                                        585
```

Finally, on line 585, we call the handler for the matched command stem.
The compiler knows how to correctly build the call frame so we can use a simple dispatch table rather than a compound if statement (which are dog slow).

4.15.2 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*.

4.15.3 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, be used with Windows. This opens up the ability to also 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** (all 16 bytes of it) 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).

4.16 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.

4.16.1 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. If the device is defective or simply not present, you will find it rather difficult to save commands.

4.16.2 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.

Although an FRAM smaller than 64Kb can be fitted to the board without knowledge of the device size, the software will not deal with an address wrap-around and happily overwrite the begining of the device when more than 256 records are stored in the device.

Thw wrap-around point moves closer to the begining as the device gets smaller. So do take note from this discussion that a smaller device will appear to work, but will behave badly should it be over-filled.

Therefore the software needs a working device in location U3 or it won't be capable of operating in a hunt.

4.16.3 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.

Byte at-a-time or two byte at-a-time devices may operate if the hex-file loader inserts appropriate inter-line delays during the load operation.

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 entire audio file system must be loaded following a device erase (sector erase is not tested as of V3.64).

4.16.4 EEPROM

Can we deal with EEPROM, or do we even need to think about this type of device?

For this discussion, I will consider the STMicroelectronics M95P08. This is an 8Mb device in an 8 pin SOIC package.

Page Size is 512 bytes, so we are within limits for out 32 byte logicl page used in the operating software.

This supported SPI modes appear to be compatible with the FRAM and FLASH devices we are currently using.

The status register has the **WIP** (Write In Progress) bit to test for programming completion.

The device supports several bulk erase commands.

The device supports the (**FLASH_COMMAND_WRITE** 0x02) command. This writes one to 512 bytes. The write block must be page aligned (all FLASH require this).

This performs an erase operation on the affected bytes before the program step is performed.

This *feels* like the FRAM but with a slow write cycle time.

The device supports the (**FLASH_COMMAND1_JEDECID** 0x9F) command. It returns a standard code (0x20, 0x00, 0x14).

The device **does not** support the (**FLASH_COMMAND4_RDID** 0x90) command.

It appears this device should work with an update to the device table.

So, in the V4.11 release, we add an EEPROM device that decodes like a FLASH device. It will show up as **EEPROM** from the command line, but is handled jsut like a FLASH device internally.

It will respond to an erase command, but this isn't strictly necessary.

This provides for a large audio file system for the older 102-73161-25 boards while still being able to quickly update the commands.

The audio file system must start **after** the area reserved for commands. Nominally, you will reserve one block (64K) for commands.

4.16.5 FLASH and FRAM JEDEC IDs

There is a somewhat generous device recgonition table built into the base software that recgonizes many FRAM and FLASH devices.

Listing 4.8: JEDEC table

```
sts41,00*     Write-Mode JEDEC-ID  Size  Page  Sctr     Manufact      Device    1
sts41,00*     FLASH_PAGE 20.8A.17  64M   256     64K      Micron     MT25QL64    2
sts41,01*     FLASH_PAGE EF.70.17  64M   256     64K     Winbond     W25Q64JV    3
sts41,02*     FLASH_PAGE 20.71.17  64M   256     64K      Micron     M25PX64     4
```

Devices 64Mb and above.

The **FLASH_PAGE32** indicates that these devices require a 32 bit address. They operate just like all the other **FLASH_PAGE**, other than the added address byte.

Listing 4.9: JEDEC table 5

```
sts41,03*    FLASH_PAGE 20.71.16  32M  256  64K      Micron  M25PX32    5
sts41,04*    FLASH_PAGE 9D.13.46  32M  256  64K        ISSI  25CQ032    6
sts41,05*    FLASH_PAGE 68.40.16  32M  256  64K       BYTe  BY25Q32    7
sts41,06*    FLASH_PAGE 1F.86.01  16M  256  64K     Renesas  AT25SF16   8
sts41,07*    FLASH_PAGE C2.23.15  16M  256  64K    Macronix  MX25V16    9
sts41,08*    FLASH_PAGE C2.20.15  16M  256  64K    Macronix  MX25V16   10
sts41,09*    FLASH_PAGE 20.71.15  16M  256  64K      Micron  M25PX16   11
sts41,10*    FLASH_PAGE 9D.13.45  16M  256  64K        ISSI  25CQ016   12
sts41,11*    FLASH_PAGE 68.40.15  16M  256  64K       BYTe  BY25Q16   13
sts41,12*    FLASH_PAGE 20.71.14   8M  256  64K      Micron  M25PX80   14
sts41,13*    FLASH_FRAM C2.2E.03   8M    1   0K     Cypress  CY15B108Q 15
sts41,14*    FLASH_PAGE 20.71.14   8M  256  64K      Micron  M25PX80   16
sts41,15*    FLASH_PAGE 9D.13.44   8M  256  64K        ISSI  25LQ080   17
sts41,16*    FLASH_PAGE 68.40.14   8M  256  64K       BYTe  BY25Q80   18
sts41,17*    FLASH_FRAM C2.26.08   4M    1   0K     Cypress  CY15B104Q 19
sts41,18*    FLASH_FRAM 04.49.03   4M    1   0K     Fujitsu  MB85RS4MT 20
sts41,19*    FLASH_PAGE 9D.7F.7E   4M  256  64K        ISSI  25LD040   21
sts41,20*    FLASH_PAGE 9D.7E.7F   4M  256  64K        ISSI  25LD040   22
sts41,21*    FLASH_PAGE 9D.7E.FF   4M  256  64K        ISSI  25LD040   23
```

Devices 4Mb to 32Mb.

The **FLASH_FRAM** devices at this end of the list are generally quite expensive.

Avoid using the **FLASH_AAI** devices, they may not provide adequate write performance, particularly with the binary loader.

The **FLASH_PAGE** device toward the top of the list provide generous space for audio. The 4Mb devices provide over 2 minutes of audio at the standard 4KHz sample rate.

Listing 4.10: JEDEC table 24

```
sts41,22*    FLASH_FRAM C2.2A.64   2M   1   0K   Cypress   CY15B102Q   24
sts41,23*    FLASH_FRAM C2.2A.60   2M   1   0K   Cypress   CY15B102Q   25
sts41,24*    FLASH_FRAM C2.2A.04   2M   1   0K   Cypress   CY15V102Q   26
sts41,25*    FLASH_FRAM C2.2A.00   2M   1   0K   Cypress   CY15B102Q   27
sts41,26*    FLASH_FRAM 04.48.03   2M   1   0K   Fujitsu   MB85RS2MTA  28
sts41,27*    FLASH_FRAM C2.25.08   2M   1   0K   Cypress   FM25V20A    29
sts41,28*    FLASH_FRAM C2.25.00   2M   1   0K   Cypress   FM25V20     30
sts41,29*    FLASH_PAGE 7F.9D.22   2M   256 64K    ISSI     25LD020     31
sts41,30*    FLASH_FRAM C2.24.00   1M   1   0K   Cypress   FM25V10     32
sts41,31*    FLASH_FRAM C2.24.01   1M   1   0K   Cypress   FM25VN10    33
sts41,32*    FLASH_FRAM AE.83.09   1M   1   0K   Lapis     MR45V100A   34
sts41,33*    FLASH_PAGE 7F.9D.21   1M   256 4K     ISSI     25CD010     35
sts41,34*    FLASH_FRAM 04.27.03   1M   1   0K   Fujitsu   MB85RS1MT   36
sts41,35*    FLASH_FRAM 7F.27.03   1M   1   0K   Fujitsu   MB85RS1MT   37
sts41,36*    FLASH_FRAM C2.23.00   512K 1   0K   Cypress   FM25V05     38
sts41,37*    FLASH_FRAM 04.26.03   512K 1   0K   Fujitsu   MB85RS512T  39
sts41,38*    FLASH_FRAM 7F.26.03   512K 1   0K   Fujitsu   MB85RS512T  40
sts41,39*    FLASH_PAGE 7F.9D.20   512K 256 4K     ISSI     25CD512     41
```

Devices 512Kb to 2Mb.

The **FLASH_FRAM** devices in the middle of the list will still be expensive.

Do not use the **FLASH_AAI** devices, they are deprecated as they didn't provide adequate write performance as they program 2 bytes at a time. These presented a problem with the binary loader.

Listing 4.11: JEDEC table 42

```
sts41,40*    FLASH_FRAM C2.22.C8   256K 1   0K   Cypress   CY15B256Q   42
sts41,41*    FLASH_FRAM C2.22.08   256K 1   0K   Cypress   FM25V02A    43
sts41,42*    FLASH_FRAM 04.25.03   256K 1   0K   Fujitsu   MB85RS256TY 44
sts41,43*    FLASH_FRAM C2.21.08   128K 1   0K   Cypress   FM25V01A    45
sts41,44*    FLASH_FRAM C2.21.88   128K 1   0K   Cypress   CY15B128Q   46
sts41,45*    FLASH_FRAM 04.03.02   64K  1   0K   Fujitsu   MB85RS64V   47
sts41,46*    FLASH_FRAM 04.7F.23   64K  1   0K   Fujitsu   MB85RS64T   48
sts41,47*    FLASH_FRAM C2.23.00   64K  1   0K   Cypress   FM25V05     49
sts41,48*    FLASH_FRAM FF.FF.FF   64K  1   0K   Unknown   64K FRAM    50
```

Devices less than 512Kb.

The **FLASH_FRAM** devices at this end of the list start to become much lower in price.

You will probably find the 128K and 256K FRAM devices much more reasonably priced while providing a reasonably large space for storing commands.

Take special note of the device on line 55, the default device.

eThis is the assumed configuration when the devices does not respond with a valid **JEDEC ID** when it is queried. Some FRAM devices lack the **JEDEC ID** device commands so they will be enumerated as a 64Kb device.

This provides space for 256 commands which is typically adequate for setting up any fox hunt. Larger devices allow for storing more complex or multiple hunt scenarios.

4.17 RF Daughterboards

Discussions of the RF amplifiers.

4.17.1 102-73161-22: Bypass

A simple amplifier bypass card.

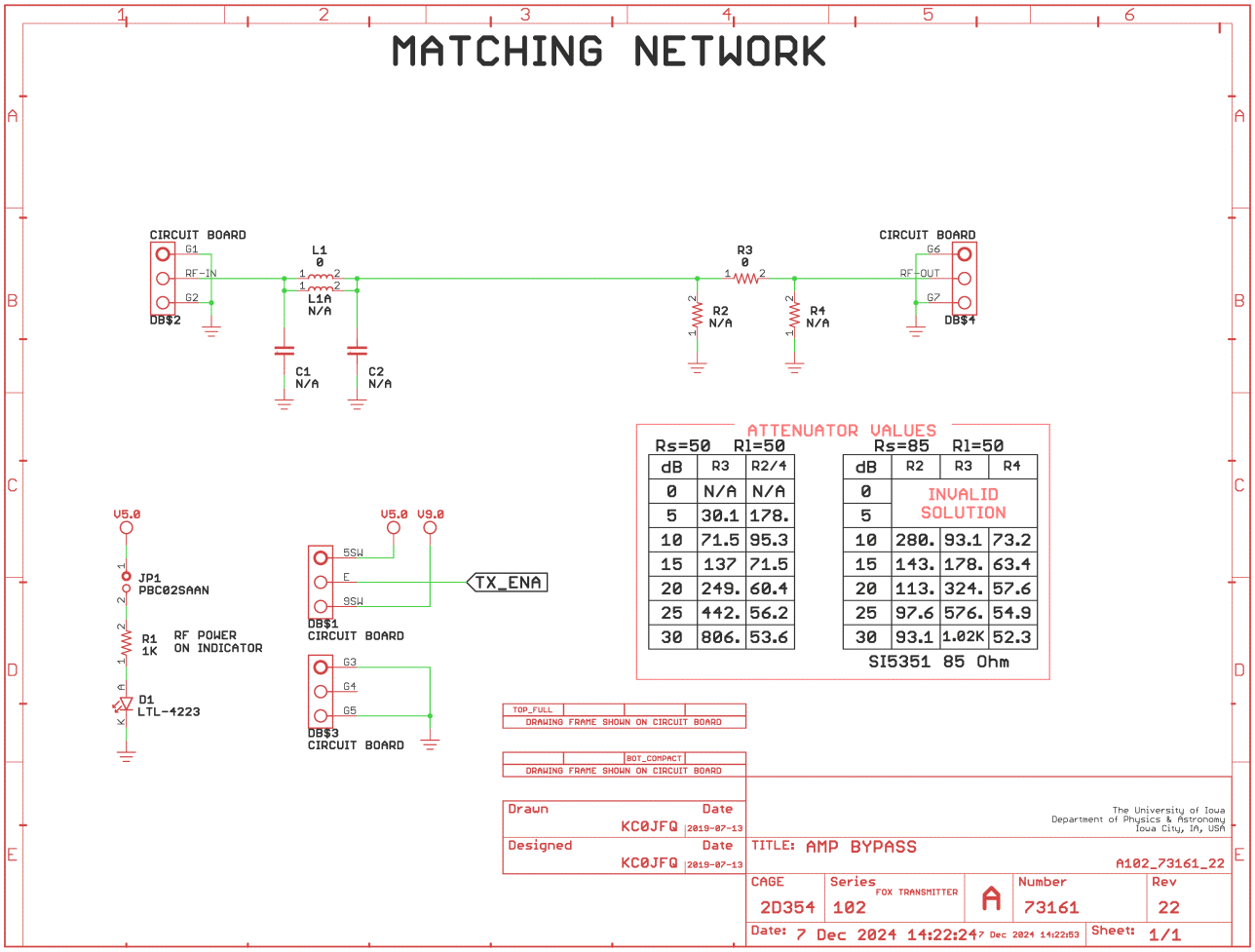


Figure 4.30: Amplifier Bypass Schematic

This is used to feed the synthesizer clock directly through to the LPF filter on the main board. There are provision for imnpedance matching and attenuation on the board.

4.17.2 102-73161-24: Class-C

A simple Class-C amplifier.

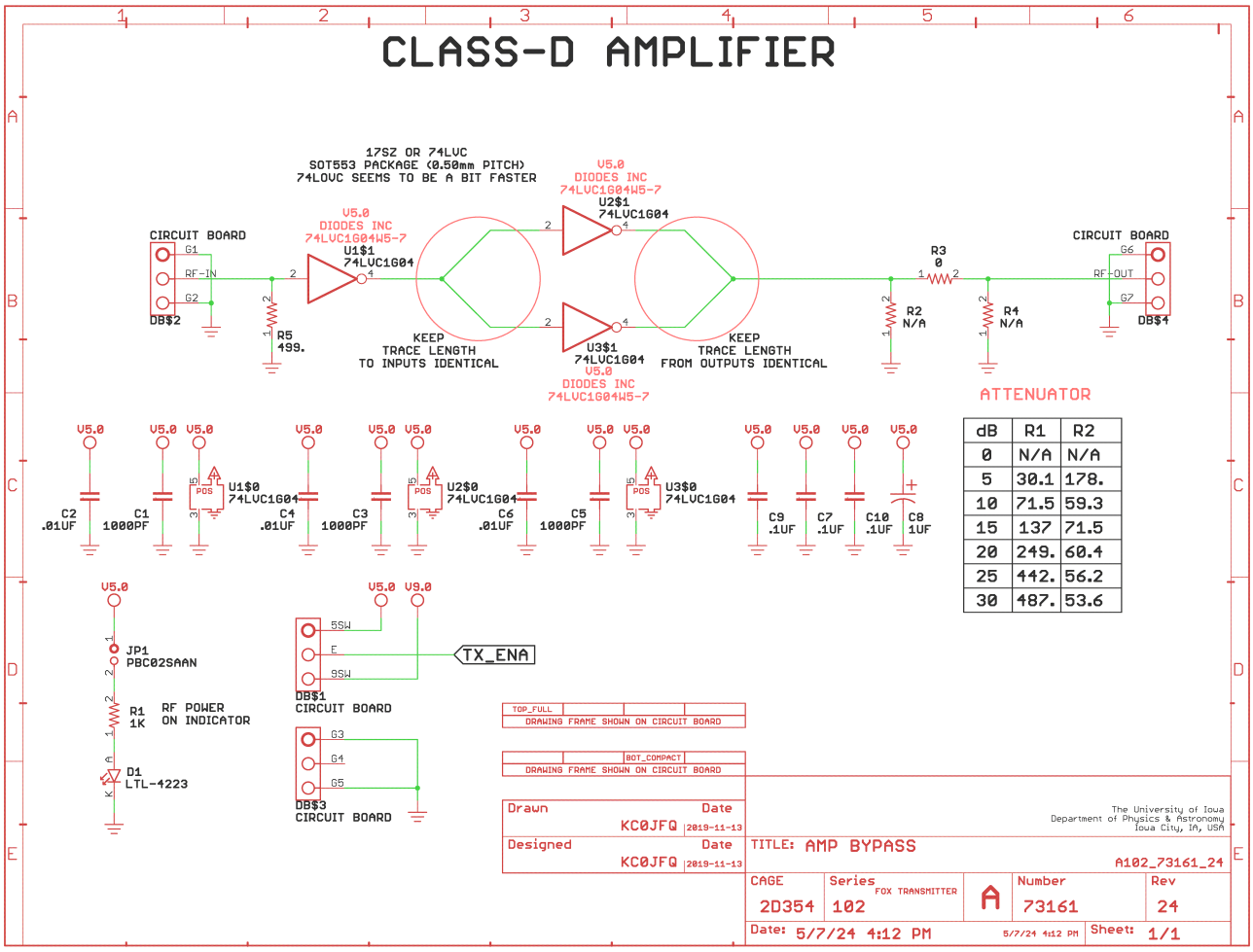


Figure 4.31: Class-C Amplifier Schematic

This power amplifier board uses simple CMOS gates to provide current gain (above what the ICS525/SI5351 can supply) 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.

Take note of the power connections for the gates on this board. The 74LVC1G04 is powered from the 5V rail. The SI5351 will probably not provide adequate drive levels for this amplifier to work well.

4.17.3 102-73161-29 LVDS Class-C

74LVC Power Amplifier Daughter board.

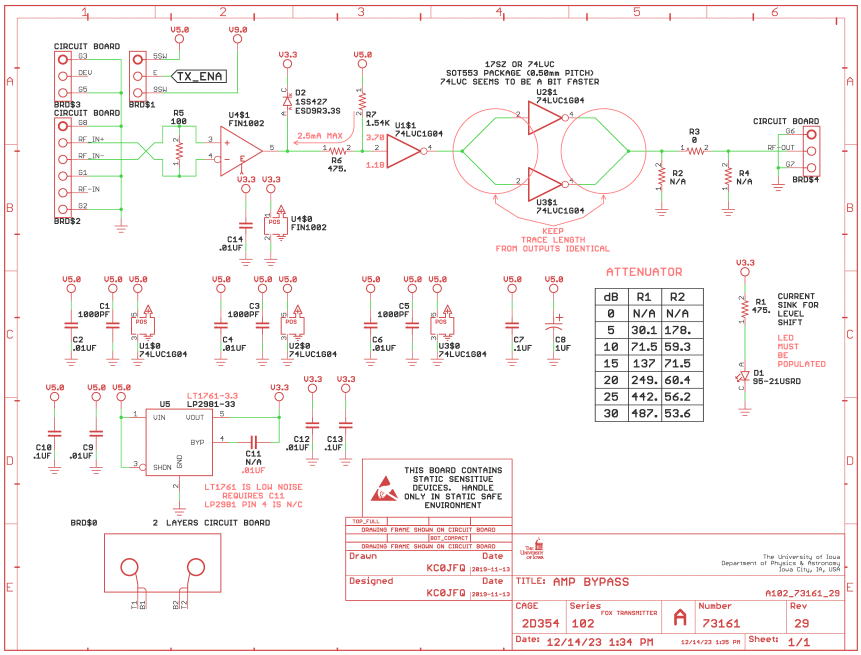


Figure 4.32: LVDS Amplifier Schematic

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).

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.

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.

Observe that the RF carrier signal, generated by the SI5351, arrives as a differential signal on the connector. Also this differential signal has a pair of dedicated pins. This requires that the motherboard provide the RF carrier signal as a differential signal. This being effected by a FIN1001 device on the motherboard (U11).

It should be obvious that U11 on the motherboard must be populated when using an RF daughterboard with a FIN1002 receiver. The system will happily configure for operation with this daughterboard and produce no putput should U11 be left off the motherboard!

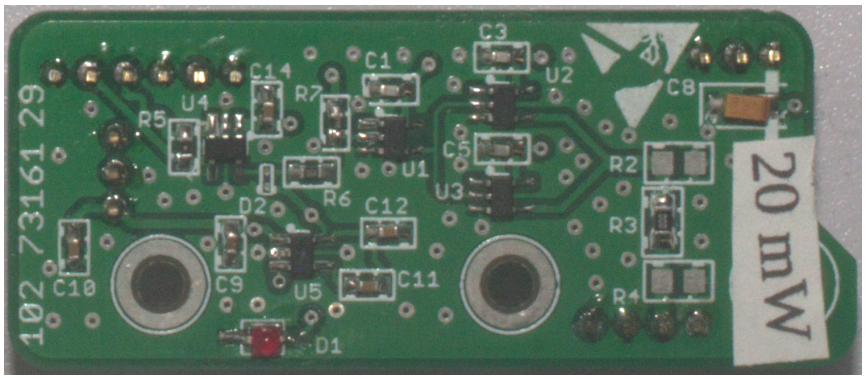


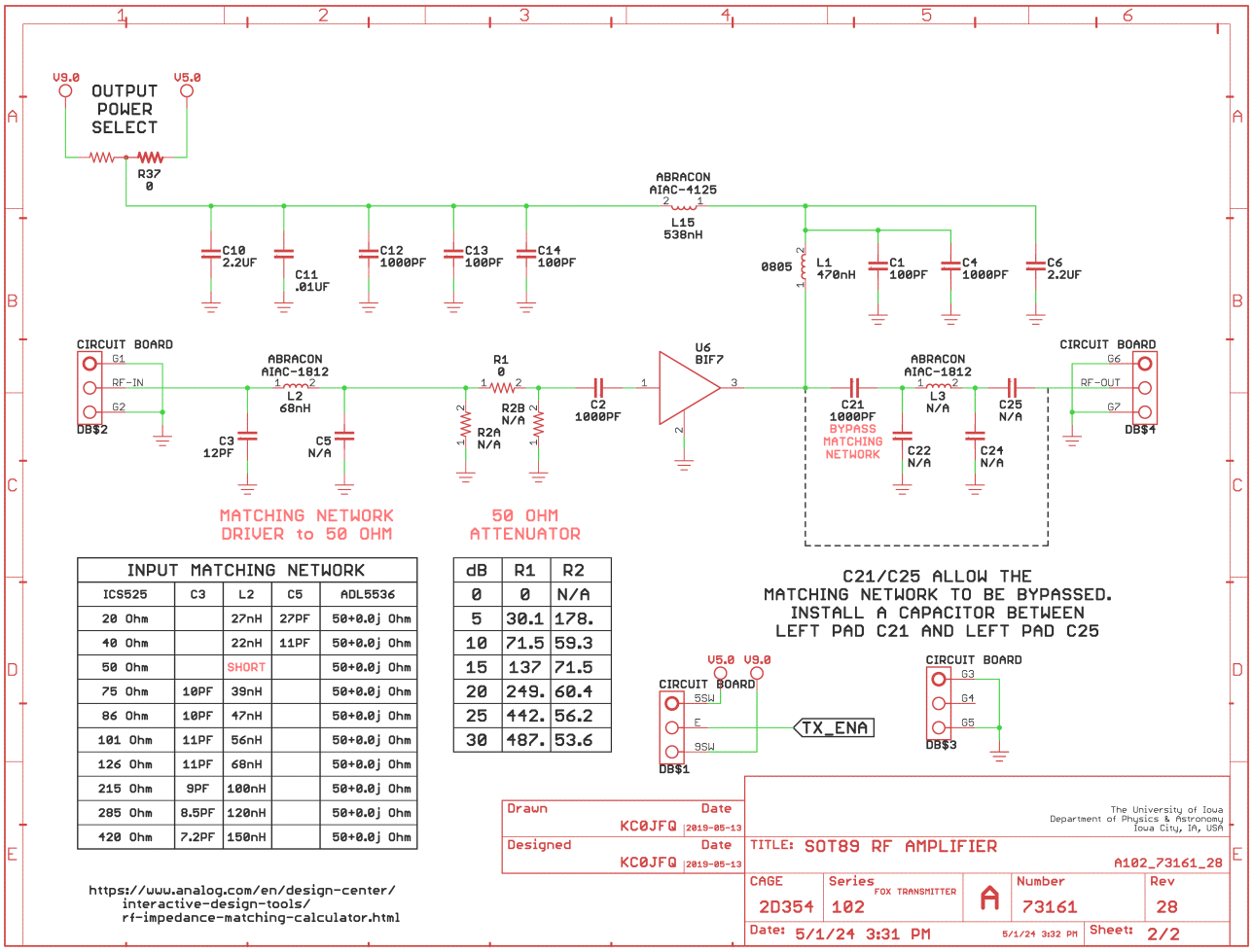
Figure 4.33: LVDS Amplifier

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.

4.17.4 102-73161-28: MMIC

MMIC Power Amplifier Daughter board.



INPUT MATCHING NETWORK				
ICS525	C3	L2	C5	ADL5536
20 Ohm		27nH	27PF	50+0.0j Ohm
40 Ohm		22nH	11PF	50+0.0j Ohm
50 Ohm		SHORT		50+0.0j Ohm
75 Ohm	10PF	39nH		50+0.0j Ohm
86 Ohm	10PF	47nH		50+0.0j Ohm
101 Ohm	11PF	56nH		50+0.0j Ohm
126 Ohm	11PF	68nH		50+0.0j Ohm
215 Ohm	9PF	100nH		50+0.0j Ohm
285 Ohm	8.5PF	120nH		50+0.0j Ohm
420 Ohm	7.2PF	150nH		50+0.0j Ohm

dB	R1	R2
0	0	N/A
5	30.1	178.
10	71.5	59.3
15	137	71.5
20	249.	60.4
25	442.	56.2
30	487.	53.6

Figure 4.34: Class C amplifier using ADL5536 or similar.

This power amplifier board makes use of a variety of MMIC amplifiers that appear in the SOT89 package. Analog Devices provides several examples, such as the ADL5536 and ADL5544. The SI5351 provides adequate drive for this class of devices to produce adequate output levels (use the **CONF CLK0** to select the unbuffered clock).

C3, C5, and L2 form a matching network that may be used to match the incoming signal (**RF-IN**) to the 50Ω input of U6, although the SI5351 avoids having to populate the matching network. You will notice the schematic shows values for these parts, which is incorrect for the SI5351 where C3 and C5 are not populated and L2 is installed as a short.

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. Output levels can better be selected by choosing a device that produces the desired output level.

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

The output matching network is not needed as the nominal output impedance of all of this type of amplifiers are 50Ω, so C21 is installed across one pad of C21 and one pad of C25 to bypass the network.

The *Be Rex* BG15A may be used to produce a bit less than 50mW. The *Be Rex* BIF7 may be used to produce close to 100mW.

Note that most of the available MMIC chips expect no more than 5V to be provided at the output pin to power the device. R37 must be installed in position 1-2.

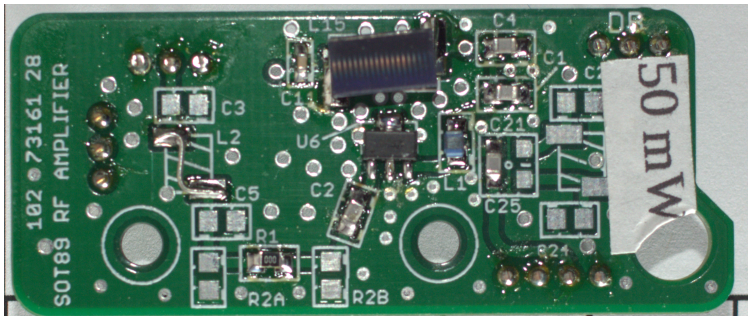


Figure 4.35: 73161-28 MMIC Amplifier Board

4.17.5 102-73181-28: MMIC Chirp

The MMIC amplifier for chirping applications.

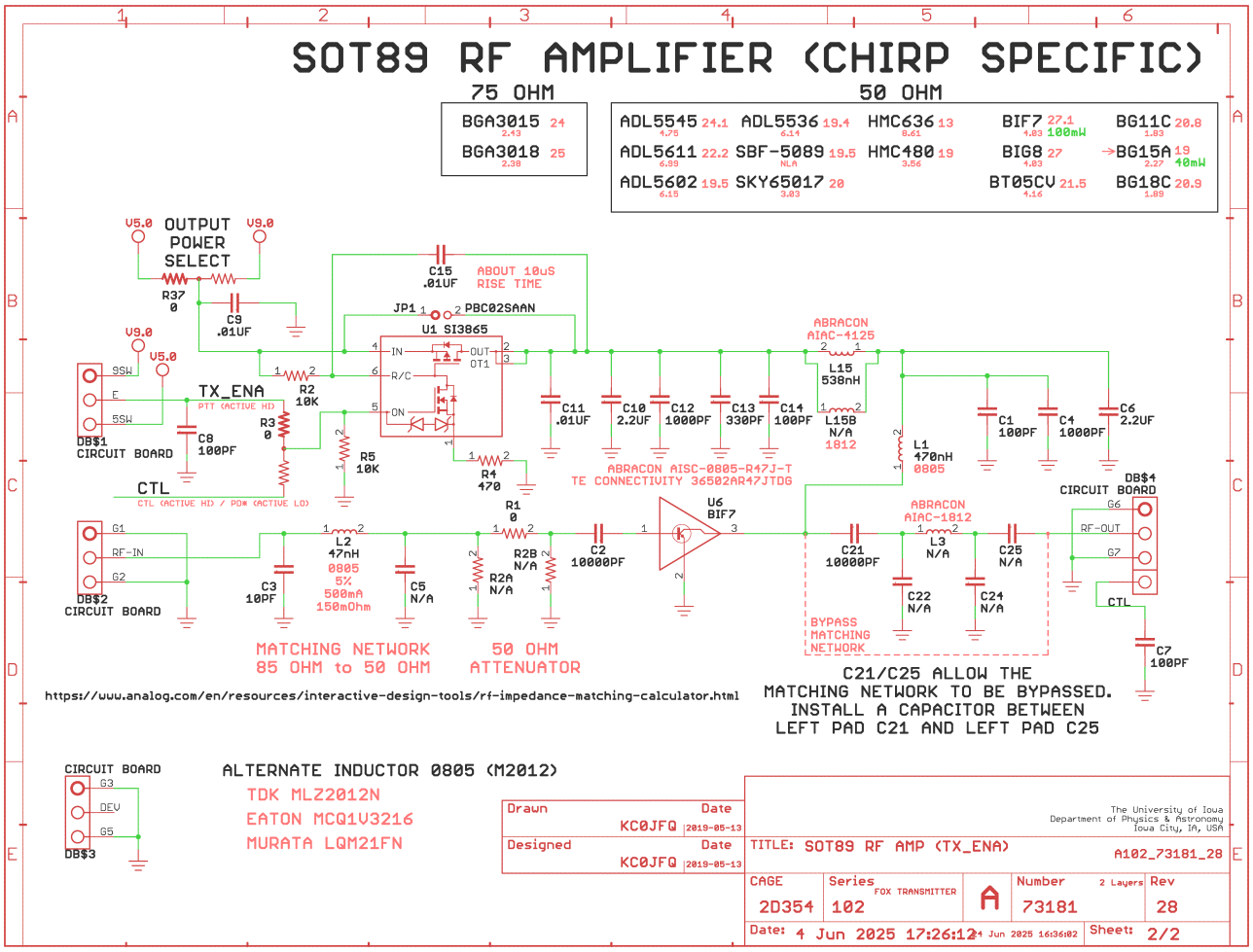


Figure 4.36: MMIC CHiRP Amplifier Schematic

This is a rework of the 102-73161-28 amplifier specifically to enable interrupted carrier operation. There are a few package selection changes for some of the inductors to make device selection a bit easier. This amplifier has been built and tested to produce between 50mW and 100mW (typically about 60mW).

This amplifier may be built up without the power switching function making it functionally identical to the 102-73161-28 amplifier. Installing JP1 will bypass the power switch.

Mechanically, this board is larger than the simple RF ampifiers. It requires the third (smaller) mounting standoff (like the 102_73181_36 DRA818/SA818 board).

This rework adds the 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 and this board will function normally.

The default position of R3 is connecting the **TX_ENA** net to the control pin (U1-5) on the power switch. This is the $PTT*$ (push-to-talk) signal

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

The updated schematic uses values of 10KΩ, 470Ω, and .01uF to achieve a rise time of 10uSec.

These values keeps L15 from ringing at turn-on.

Take note of the fact that the switch simply removes power from the amplifier (**U6**) but does not, of course, attenuate the incoming RF signal. This low level carrier (below 1mW) will still *escape* the enclosure as long as the SI5351 is generating a clock (i.e. until the **DONE** command occurs).

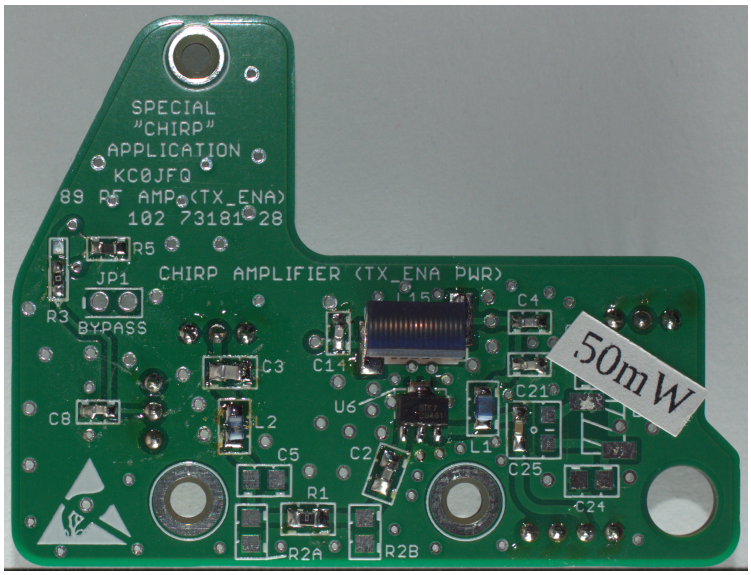


Figure 4.37: MMIC Amplifier Board

When a receiver is close to the fox transmitter, it can detect the residual carrier should the receiver squelch control be set to near open.

Removing power from the amplifier (**U6**) does reduce power draw considerably, thereby extending battery life.

To eliminate the on-boarxd switching function, do not populate the following: U1, R2, R3, R4, R5, C8, and C15.

Install a wire jumper in position JP1.

With this alteration, the board functions exactly like the 102-73161-28 amplifier board. You will notice that this schematic shows values for C5 and L2 which is incorrect for the SI5351, L2 shold be installed as a short. This assumes, of course, we are dealing with a 50Ω system.

Note that all of the 50Ω MMIC devices shown on the schematic expect no more than 5V to be provided to power the device. R37 must be installed in position 1-2.

There are a few 75Ω MMIC amplifiers that operate at higher voltages, nominally 8.4V, which is within the operating voltage of a 6-cell pack. Can we produce higher output power using these amplfiers?

INPUT MATCHING 50 Ohm MMIC				
INPUT	C3	L2	C5	MMIC AMP 50 OHM
20 Ohm		27nH	27PF	50+0.0j Ohm
40 Ohm		22nH	11PF	50+0.0j Ohm
50 Ohm		SHORT		50+0.0j Ohm
75 Ohm	10PF	39nH		50+0.0j Ohm
86 Ohm	10PF	47nH		50+0.0j Ohm
101 Ohm	11PF	56nH		50+0.0j Ohm
126 Ohm	11PF	68nH		50+0.0j Ohm
215 Ohm	9PF	100nH		50+0.0j Ohm
285 Ohm	8.5PF	120nH		50+0.0j Ohm
420 Ohm	7.2PF	150nH		50+0.0j Ohm

Figure 4.38: Matching network values 50Ω

INPUT MATCHING 75 Ohm MMIC				
INPUT	C3	L2	C5	MMIC AMP 50 OHM
20 Ohm				75+0.0j Ohm
40 Ohm				75+0.0j Ohm
50 Ohm		39nH	10PF	75+0.0j Ohm
75 Ohm		SHORT		75+0.0j Ohm
85 Ohm	4.7PF	30nH		75+0.0j Ohm

Figure 4.39: Matching network values 75Ω

The SI5351 datasheet indicates that when the output drive strength is set to 8mA, the output impedance is abount 85Ω. There is an impedance matching network of the 102-73181-28 board that we can use to transform the SI5351 output impedance to the 50Ω expected by the MMIC amplifier. This should allow the SI5351 to provide enough drive to be able to produce about one hundred milliwatts.

Both the 102-73161-28 and then 102-73181-28 boards have pads for matching networks on either side of the MMIC. The 102-73181-28 board is derived from the 102-73161-28 design so the reference designators match. The matching network discussion applies to both boards.

Assume the SI5351 output is configured with 8mA drive to produce a 50Ω output impedance. We will need to match the 75Ω input impedance of the MMIC (which is a 75Ω CATV amplifier). This is accomplished by populating the L2 position with a 39nH inductor and the C5 position with a 10pF capacitor.

The output filter on the motherboard assumes 50Ω which should match up with a 50Ω *rubber ducky* antenna. We probably do not want to change parts on the motherboard as this makes it incompatible with other amplifiers.

We do have another matching network (C22, C24, and L3) at the output of the MMIC that we can use. Here we use the same matching network, but in reverse. Populate the L3 position with a 39nH inductor and the C22 position with a 10pF capacitor. C21 will have a 0Ω resistor and C25 can then be populated with the 10,000pF blocking capacitor. We place the blocking capacitor after the matching network so that L3 is never floating.

The power selection resistor, R37, is installed in the 2-3 position to use the **V9.0** net for power. This is raw battery voltage, albeit switched, from the motherboard. We also need to verify that U91, C93, C94, R94, C96 and C97 are populated on the motherboard to supply battery voltage to the RF board.

4.17.6 102-73181-71: MMIC/MMIC Chirp

The higher power MMIC amplifier.

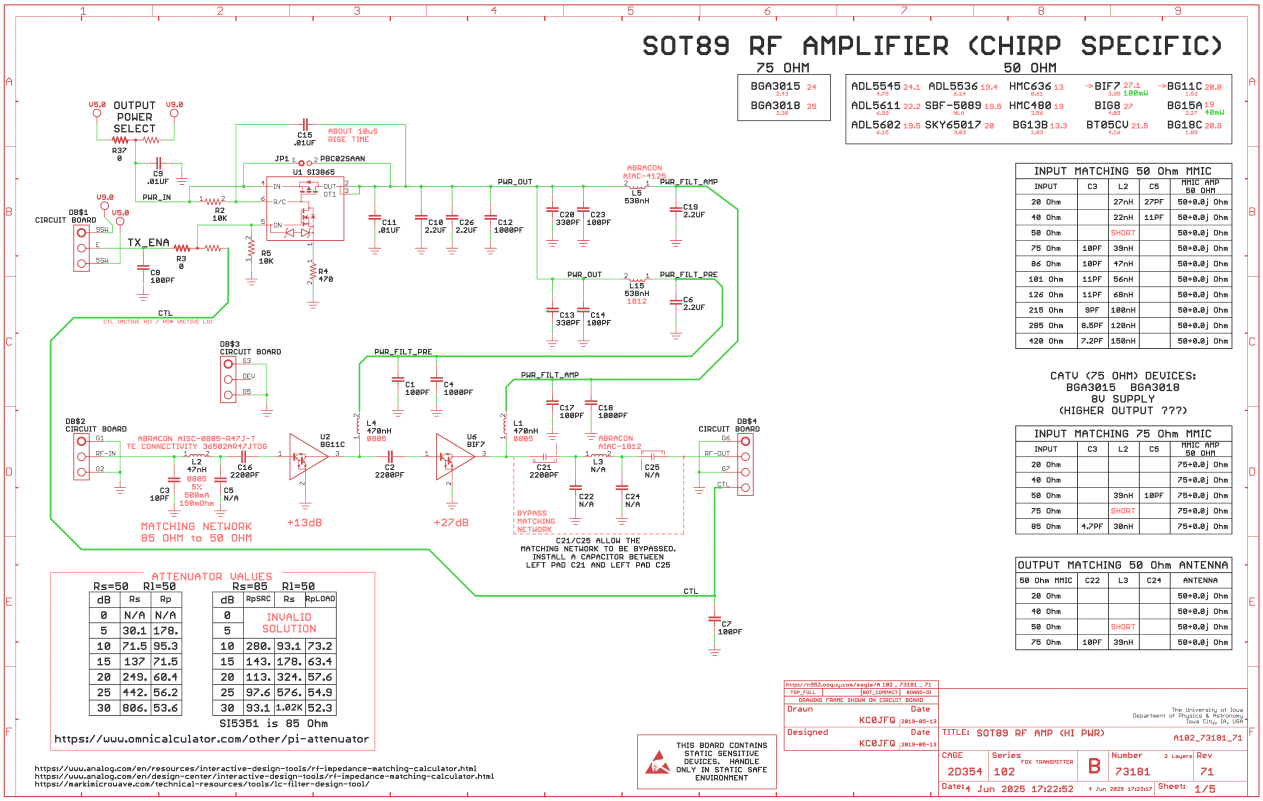


Figure 4.40: Cascaded MMIC Amplifiers

Yet another rework, this time an update to the 102-73181-28 amplifier to add a second stage to get a bit more output power. This amplifier has been tested to produce about 135mW.

This amplifier may be built up without the power switching function making it functionally identical to the 102-73161-28 amplifier.

Mechanically, this board is a bit larger than the 102-73181-28 board. It also requires the third (smaller) mounting standoff

This rework adds a second amplifier stage to provide a bit more drive to the BIF7. With the same power switching, we emulate the operation of the DRA818 daughterboard where the PTT (push-to-talk) control is seperate from the PD (power down) control.

As with the 102-73181-28 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 and this board will function normally.

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

The soft-start network, consisting of R2, R4, and C15, is present with the same function. This circuit keeps L5 and L15 from ringing at turn-on.

As with 102-73181-28 the switch (U1) simply removes power from the amplifiers ((U2) and **U6**) but does not, of course, attenuate the incoming RF signal. The attenuator seeen on the 102-73181-28 is no longer present.

Removing power from the amplifier ((U2) and **U6**) dramatically reduces power as we are dealing with two amplifier stages.

The switching function is eliminated in the same manner as on the 102-73181-28 boards.

The 102-73181-71 boards keeps the pads for matching networks on the input side of the first MMIC and on the output side of the second MMIC.

The power selection resistor, R37, is installed in the 2-3 position, same as on the 102-73181-28 board.

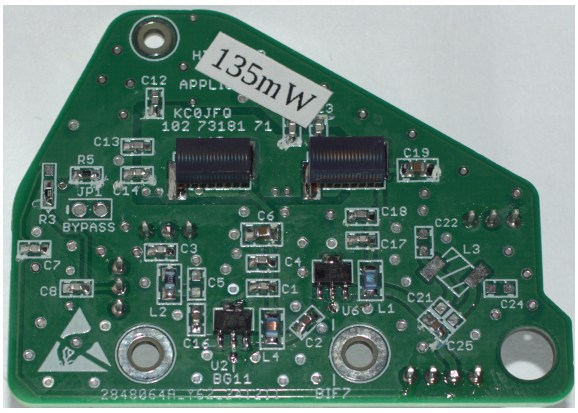


Figure 4.41: HI Power CHiRP

The circuit board.

4.17.7 102-73181-86 HI Power Bipolar CHiRP

The MAX2602 amplifier (two sheets).

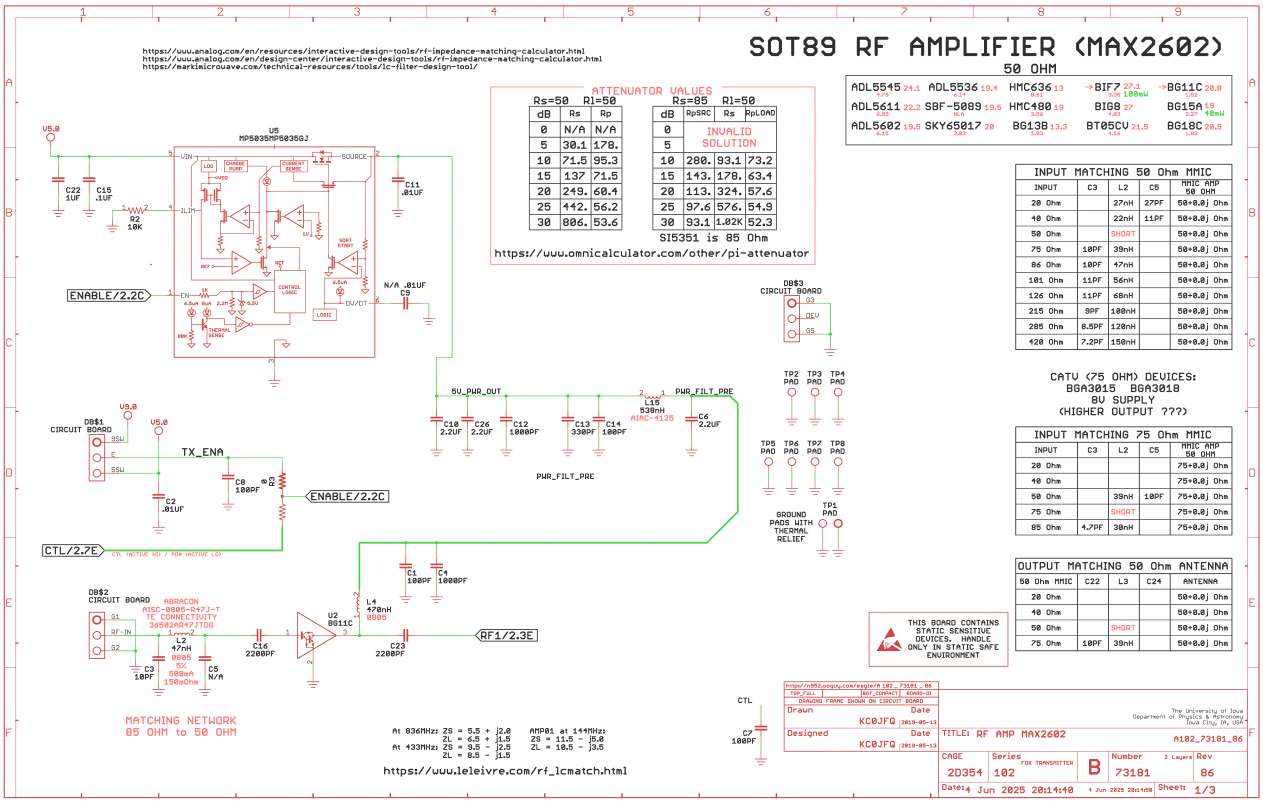


Figure 4.42: HI Power Bipolar, Sheet 1

First stage, we use one of the standard low gain MMIC IF amplifiers to boost the level from thee SI5351.

This is the second revision of this design. We switch over to the MP5035 switch to make use of the over current feature.

The power to the first stage amplifier is filtered to keep RF out of the power supply.

Input matching assumes that the SI5351 is configured for maximum drive and presents with a source impedance of 85 ohms.

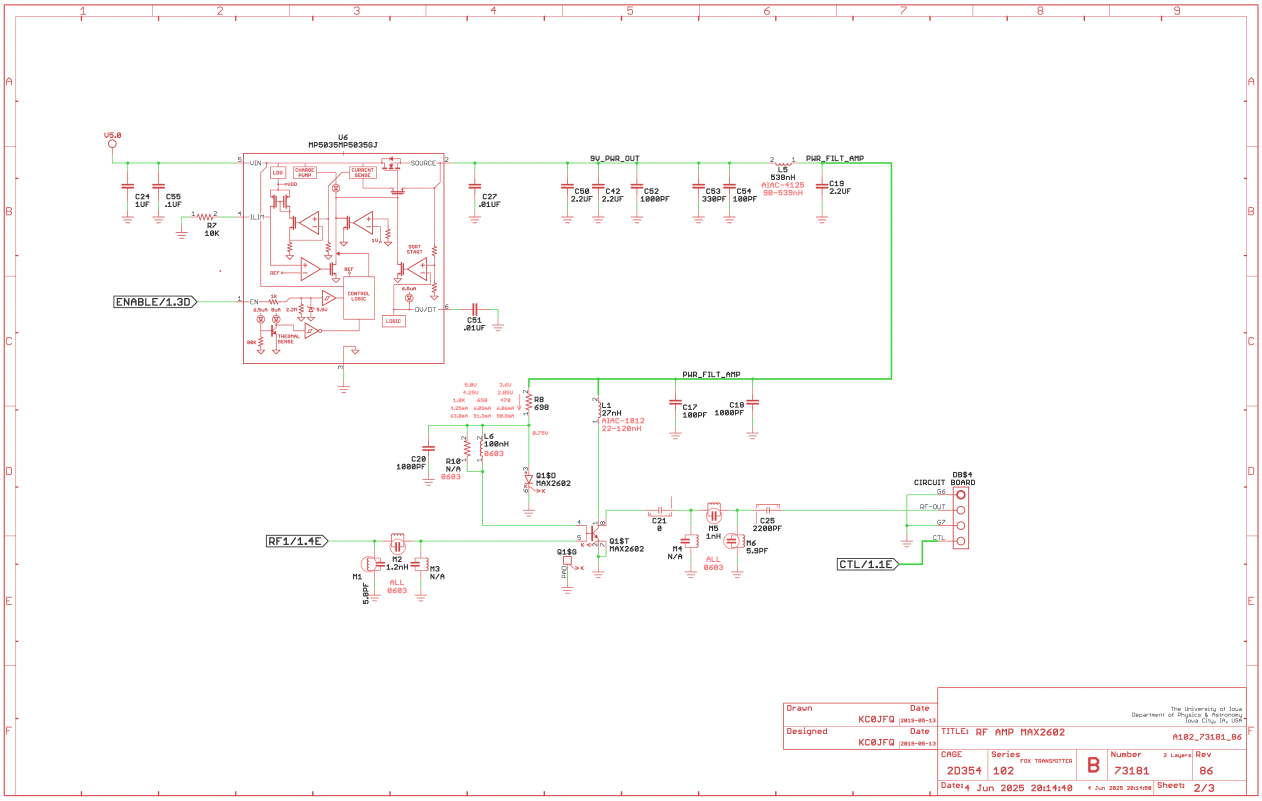


Figure 4.43: HI Power Bipolar, Sheet 2

We are now on to the MAX2602 section.

We use a second switch here, in this instance to protect the system from MAX2602 overcurrent. Filtering from the first stage to keep RF out of the supply.

M1/M2/M3 form a matching network to transform the 50Ω output of the previous stage to match the input of the MAX2602. The datasheet doesn't provide and estimate of the input impedance at this frequency.

Note that the signal is DC isolated at this point to allow biasing to come from the bias network formed from the diode in the MAX2602 package.

The bias network for the MAX2602 is formed around a diode (Q1$D) inside the MAX2602 that is process matched and isothermal to the switching transistor (Q1$T) This network should keep the base of Q1$T on the edge of conduction. The input signal, being DC isolated, should then allow Q1$T and L1 top operate in Class-C mode.

M4/M5/M6 for the output matching network. Like the input network the parts are all surface mount 0603 that can be populated with a small inductor, a small capacitor, or left open.

The output from the amplifier is DC biased. THis DC bias must be eliminated in the output filter.

Well, it seems the author just can't sit still.

We continue on the quest for more RF output!

Mechanically, this board is a bit larger than the 102-73181-75 board. It will also require the third (smaller) mounting standoff.

In this iteration we will bring back the MAX2602 (Q1$T), a bipolar transistor with a bias diode on the same substrate (Q1$D).

We add a second power switch to deal with the expected load presented by the MAX2602. The value of C55, larger than C15, ramps up power to the MAX2602 slower than the BG11C.

Q1$D is forward biased (through R10/L6) such that the transistor (Q1$T) is just starting to conduct. Any input signal will drive Q1$T into conduction. The anode of Q1$D is bypassed with a 1000PF cap.

The MAX2602 is not matched to 50Ω like the previous MMIC amplifier designs, so we require matching networks on both the input and output. The peculier synbols used in the matchingt network allow for quick changes in the network topology, that is switching C and L parts.

The assembly drawing, 102_73181_85_pwa.pdf, shows trace cuts to allow powering the MAX2602 separately from the the fox transmitter for development work.

If the MAX2602 is not correctly driven, it can draw excessive current that will pop the protection fuse on the back of the Fox Transmitter motherboard.

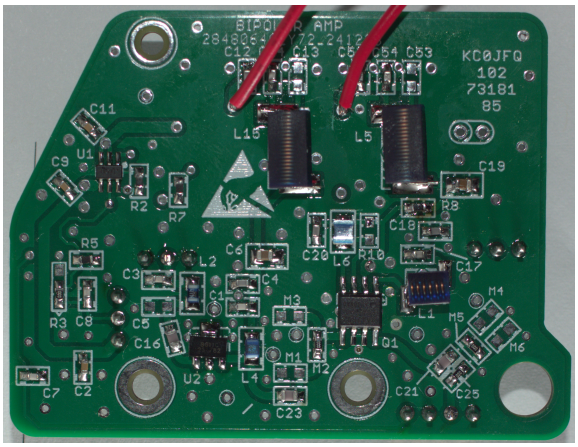


Figure 4.44: HI Power Bipolar CHiRP (102-73181-85)

The prototype board contuinued using the SI3865 power switch. The 102-73181-86 update switches to the MP5035 to take advantage of the current limit feature.

4.17.8 102-73227-11: 80M Band

It is possible to operate in the 80M band using the 102-73227-11 RF amplifier daughterboard. This daughterboard has a simple class-C amplifier and a low pass filter appropriate for the 80M band.

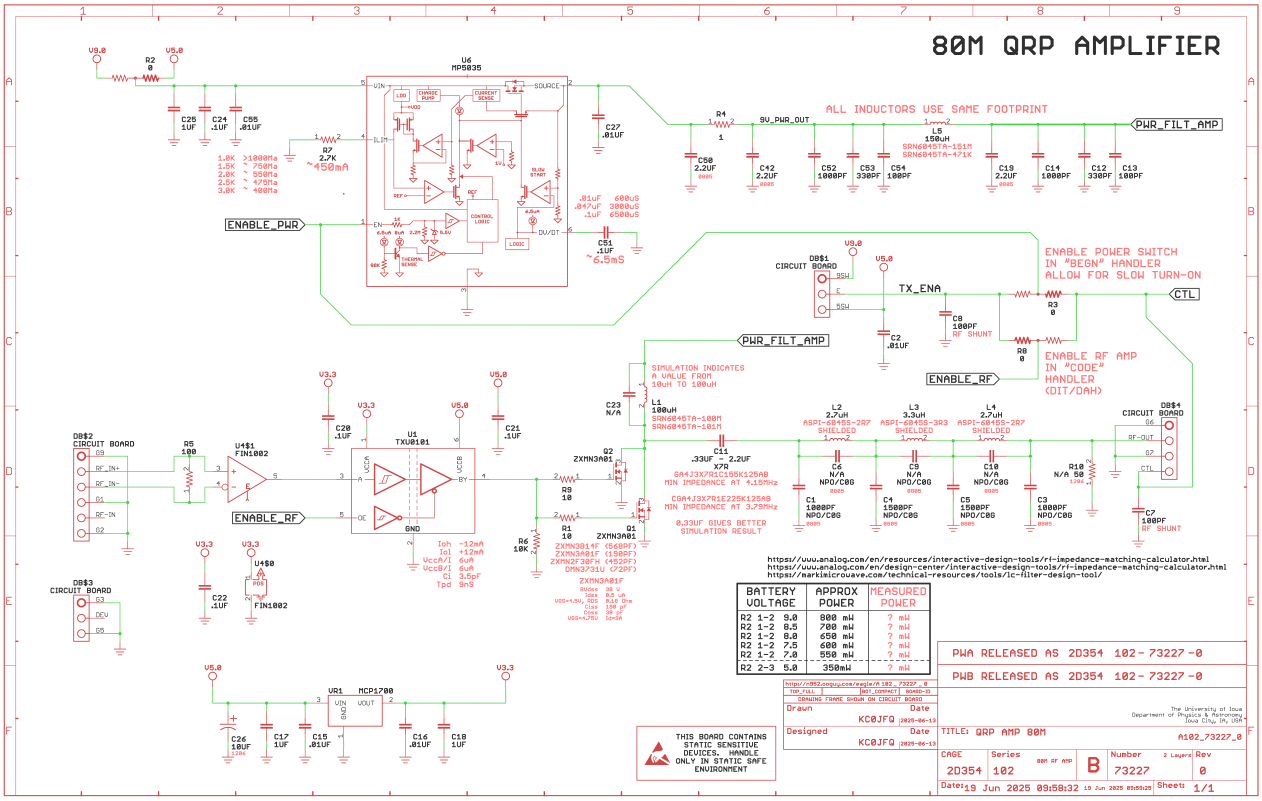


Figure 4.45: HF Amplifier and LPF

Start with the power switch.

Here we change from the SI3865 to the indicated MP5035. This change adds overcurrent protection (which was needed in the 102-73181-86 design) as well as reducing the complexity of the soft-start feature.

We add filtering on the output of the power switch (U6) in much the smae manner as those designs using the SI3865. All of the inductors use the same package, so some care must be exercised when assembling the boards to place the them correctly.

The input section accommodates both the older ICS525 design as well and the current SI53512 design.

The SI5351 can directly generate carrier down in the lower bands, so we take a direct input through the LVDS receiver (U4). This assumes, of course, that the motherboard has the corresponding LVDS driver installed.

The SI5351 can be programmed to operate at any target frequency that you wish to operate at.

The ICS525, on the other hand, is a much more limiting part. It is incapable of operating below 30MHz, so we need the binary counter (U2) to provide a means of allowing the ICS525 to provide a useful reference clock.

The ICS525 is, however, still very limiting in frequency selection is you want to operate at a particular frequency.

Drive to the switching element is provided by the level shifter, U1.

By driving Q1 and Q2 at 5 volts, the universe of parts expands slightly and we also improve the *Rds* characteristic slightly.

The basic amplifier topology is Class-C.

Note that C23 and C29 are not populated and that the value of L1 is not critical.

Simulations indicate that the current coming into L1 is constant and around 200mA.

The circuit will operate with only one MOSFET installed, the second being added to lower the *Rds* seen by the circuit.

Operating the amplifier as Class-E.

The appropriate value of L1 will need to be calculated. C23 and C29 are provided to form the resonant circuit. The location and value of the capacitor also need to be calculated.

Output Filter.

The on-board LPF allows this board to simply drop-in to a standard 102-73181-10 system without having to change filter components on the main board.

As shown, it is populated as a 7th. order Chebyshev. Unpopulated parts positions allow configuring as a 7th. order Elliptic if desired.

All inductors on the board are the same package!

This daughter board is intended to allow the base Fox Transmitter to operate down in the 80M band. The Low Pass Filter on the motherboard is transparent at these frequencies so alternate provisions are required.

Q1 and Q2 are driven by a simple logic gate.

We duplicate the LPF topology from the main board and increase the footprint for the inductors.

The power switching more-or-less functions like the DRA818/SA818 module. **Power** and **Enable** are controlled seperately.

U6 provides a *soft start* capability, rather like the 102-73181-28 and 102-73181-71 amplifiers. The ramp-up time is quite long that probably could be shortened. It does allow time for the filter capacitors downstream of C27 to charge without drawing excessive current.

The MP5035 also has an over-current detect/lockout to try to avoid popping the battery fuse on the main board.

102-73161-25 Compatibility

The 102-73227-11 board is provisioned with a divider to accommodate the limitations of the ICS525. Normally this is not populated as the SI5351 can generate a carrier clock directly for the 80M band. For operation with the ICS525, the FIN1002 (U4) is unpopulated and the 74LVC161 (U2) is installed. The 74LVC161 provides a divide-by-16 function while sourcing its input clock from the single-ended signal taken from the ICS525 (102-73161-25 does not source an LVDS signal).

The ICS525 can then operate above its 30MHz minimum by using the 74LVC161 to generate the final clock. For the ICS525, you will have to edit the external frequency table.

4.17.9 102-73181-36 DRA818 1W RF transceiver

This daughter board is the upgrade that gets if right!
The schematic is on page 39. A a board image may be found on page 40.

This final revision breaks the DRA818 **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 control from the **PTT*** net. Again, to allow the software control of the timing .
These changes finally brought the DRA818/DRA818 module to life.

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

4.18 Deprecated RF Amplifiers

RF amplifiers that are not in use:

1. **102-73161-23** SOT-89 MMIC

   (See section 4.18.4 on page 95).
   Class C amplifier using ADL5536 or similar. Part substitution required for DC block.

2. **102-73161-29** LVDS 74LVC04 gate

   (See section 4.17.3 on page 78).
   Class D amplifier using two 74LVC04 CMOS gates. Input is LVDS from SI5351.

3. **102-73181-36** DRA818/SA818 RF Module

   (See section 4.17.9 on page 94).
   RF Module

4. **102-73227-11** 80M RF Module

   (See section 4.17.8 on page 92).
   80M RF Module

4.18.1 102-73161-12S

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.

4.18.2 102-73161-12M

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.

4.18.3 102-73161-21 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.

4.18.4 102-73161-23 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.

4.18.5 102-73161-27 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.

4.18.6 102-73181-22 DRA818 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 DRA818/SA818 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.

4.18.7 102-73181-24 DRA818 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

4.18.8 102-73181-34 DRA818 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

Chapter 5

Operation

Notes on the physical operation of the FOX Transmitter.

Do NOT handle the transmitter by the antenna!

The antenna connector mounting pins are not designed to carry the load of the transmitter enclosure and battery. Handling the tranmsitter by the antenna will, eventually, result in fracturing the mounting pins of the BNC connector. (not the solder joint!)

Replacing the connector requires considerable heat which may damage the board if not done properly.

5.1 Power

The device is powered in the field using a battery.
An LR22 9V may be expected to run the unit for several hours when using a low power amplifier. 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 typically cheaper to purchase than a single LR22. The 6-cell pack will last considerably longer and allow use of a higher power RF amplifier.

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

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 allows the use of higher voltage batteries without the accompanying heat. Switching to lithium chemistry batteries would allow extended operation or operating at higher power levels.

Exercise standard care with respect to leaving batteries in the transmitter for long periods of time. A discharged primary cell tends to leak and destroy the circuit board.

Keep in mind that the main battery provides a very samll current to keep the TOY clock running and to keep the backup battery *topped off*. Units that used a removeable backup battery can have all chemical threats removed for long term storage. Although the main battery is easily replaced, the ML1220 backup is soldered in place and may need to be replaced if allowed to fully discharge.

It should also be mentioned here that the inexpensive battery holder specified is not well suited to dealing with a leaking cell. This usually results in non-conductive corrosion salts forming on the spring and contact. Please do not be shy about discarding a battery holder that seems not to work well. Although temporary measures can be employed to force the holder to work (such as rotating the cells to burnish the contacts or even sanding the contacts) this may result in loss of power during an event!

5.2 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 motherboard may also be built with tip jacks in place of the BNC connector. This 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

5.3 Jumpers

Table 5.1: Jumpers

Ref Des	**IN**	**OUT**	**Description**
JP2	*MAS=*		MASTER Jumper
JP3	*TEST=*		TEST Jumper
JP4	*USB*	*Battery*	USB Power Configuration
JP5	*Development*	*Deploy*	Code LED
JP6	*Development*	*Deploy*	Code Buzzer
JP7	(*DRA818*)	*SI5351*	5V daughter board power bypass

5.3.1 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 5.2: 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. zero or one jumper in these two positions, we run the initialization commands to establish the identity of the transmitter. Station callsign and nickname should show up only in ther **INI=** commands.

If neither jumper is installed, we proceed with the announce message. This announce message is intended to let the hunt operator know that the transmitter is functioning as it is placed for the hunt.

When one jumper is in place, the announce message (i.e. the **ANN=** commands) is replaced with the **TEST=** commands (JP3) or the **MAS=** commands (JP2).
Note that the **INI=** commands are run for both these cases to establish the station identity.

Installing both jumpers places the system is a **Fault Recovery State**.
This **skips all commands** from the FRAM to allow control of the uninitialized system from the serial port (i.e. you can erase the entire FRAM, or clear out commands that prevent proper operation).

With both jumpers installed, the station identity will not be set, so commands that send the callsign or nickname will not operate correctly. This also means the operating frequency is not set, affecting the **BEGN** command.

5.3.2 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).

5.3.3 Jumper JP5: LED

Sheet 2.8D

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

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

5.3.4 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.

With R12, R58, Q5 and BZ1 installed, you will hear code on the buzzer. This can be used to assist in debugging your operating sequence or for a benchtop demonstration of the Fox Transmitter.

5.3.5 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 DRA818/SA818 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 pin from the zNEO. This has the potential to reduce power consumption slightly by removing power from the DRA818/SA818 module when not transmitting. 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 fair bit of time to recover from a power-down.

5.4 Resistor Jumpers

Table 5.3: Resistor Selection

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

5.4.1 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.

5.4.2 Resistor Jumper: R5 & R6

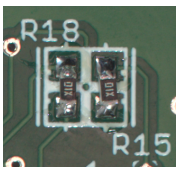
Sheet 2.4D

Selects the pin on the programming header connected to the zNEO.
Always install R5, leaving R6 unpopulated as shown on the schematic and in the parts list.

5.4.3 Resistor Jumper: R15 & R18

Sheet 3.2B

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

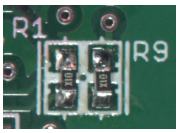


This jumper pair connects the zNEO port 1 pins to the FT232RL serial pins and to the U1 buffer for the TTL-232R-3V3-AJ cable.
There should never a need to change the orientation of these resistors.

5.4.4 Resistor Jumper: R1 & R9

Sheet 3.4D

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

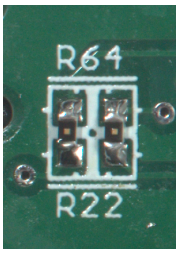


This jumper pair connects the zNEO port 0 pins to the inter-board jumpers to the DRA818/SA818 daughter board (the daughter board also has resistor positions to accomplish the same function).

5.4.5 Resistor Jumper: R64 & R22

Sheet 3.2C

102-73181-10: Install **rotated** from that indicated on the silkscreen (vertical as shown).



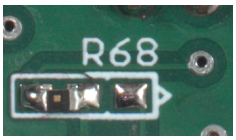
This jumper pair connects the other side of the U1 buffer to the physical connector for the TTL-232R-3V3-AJ cable.

The indicated orientation is incorrect for the TTL-232R-3V3-AJ cable, switch the resistor orientation in order to swap the Rx and Tx signals to what the TTL-232R-3V3-AJ cable requires.

5.4.6 Resistor Jumper: R68

Sheet 5.2D

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



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

Version 3.x software requires that the resistor be installed in the 2-3 position. The main board independantly manages timing for the DB_PWR signal and the TX_ENA signal. Both the DRA818/SA818 module and the chirping amplifiers require this setup.

When the resistor is installed on the 1-2 pads, the TX_ENA signal controls the power switches along with it being routed to the RF daughterboard.

In this position, the DRA818/SA818 module will not have time to complete its power-on sequence before the first setup commands are sent (and, therefore the commands are lost). The DRA818/SA818 module appears to be dead!

This position also interferes with the 102-73181-28 and 102-73181-71 amplifier modules when operating in a chirping mode.

5.5 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 typically 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.

5.5.1 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.

5.5.2 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 -SFOX2X  -t10
```

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

```
/home/wtr/Radio/halo_term/fox_simple -SFOX7  -t10 -ffox7.fox
/home/wtr/Radio/halo_term/fox_simple -SFOX2X  -t10 -ffox20.fox
```

Examples above are commands used by the author to access the 102-73161 units (FOX7) of the 102-73181 units (FOX2X). 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.

The 102-73181 units all use the cable shown in figure 4.28 on page 64. As we use one FTDI serial USB device to access all fox transmitters, we don't change the device selection supplied to **fox_simple**.

A second example is an excerpt from the shell script (bash) that was used to load an entire group of fox transmitters.

```
FOXS="/home/wtr/Radio/halo_term/fox_simple"
TACH="-SFOX2X"
FOXCMD="FOX2X_KC0JFQ.fox"
CALL="W0JV"
NAME="FOX20"
FREQ="144.225"
SCHED="300,0"
$FOXS $TACH -c250 -t10 -f$FOXCMD -lFOX2X_KC0JFQ.log \
        -C$CALL -N$NAME -R$SCHED -Q$FREQ
```

The **$FOXS**, as you would expect, is simply the Fox Station loader utility.
The **$TACH** is a keyword that determines the USB serial device we need to talk to.
The **$FOXCMD** is the common file (used for all the units in this group) with the operating schedule.

The callsign in the example is for the *Iowa City Amateur Radio Club* as this setup was for a club foxhunt.

The nickname for the group of transmitters ranged from **FOX20** through **FOX32** (big hunt!). There are several frequency groups in operation for these hunts.

The operating frequency is in the **$FREQ** variable. The announce frequency, which is the same for all units participating in the hunt, is set in the **$FOXCMD** file.

The scheduling period for this group is 5 minutes (300 seconds) and this is what is sent to the first unit in the group (offset is zero seconds).

Subsequent invocations will change only the nickname and the scheduling variable.

```
NAME="FOX21"
SCHED="300,60"
$FOXS $TACH -c250 -t10 -f$FOXCMD -lFOX2X_KC0JFQ.log \
        -C$CALL -N$NAME -R$SCHED -Q$FREQ
```

The other variables remain the same.

We end up with all five units (300 second cycle period, with 60 seconds allocated to each station) operating on the same frequency in a fully synchronized manner.

The callsign and nickname are substituted in the **$CALL** and **$NAME** commands in the fox transmitter itself. The other variables are substituted in the loader utility as the transmitter is loaded.

We can also *brute force* the time (for a single unit) by getting time from the host computer.

Unix/Linux:

```
date +"%H:%M:%S"
```

Simply copy the time value from this (or equivalent) to a **TIME** command sent to the target **fox transmitter**.

```
TIME 15:08:32
```

It bears repeating that this shortcut is effective only for loading a single unit! We need a more precise loading method (as suggested at the begining of section 5.5.2) when synchronizing multiple transmitteers.

5.6 External Frequency Tables

Work on the 102-73181-10 hardware revealed that the reference crystal used with the SI5351 may require that **CT1** and **CT2** be installed to trim the frequency of X5 such that the register values calculated assuming a 20MHz crystal will result in the desired frequency of operation.

The parts (**CT1** and **CT2**) themselves are relatively expensive (at around $3.50 each). Although you may need only one installed, an alternate approach may be used to generate the register values using the actual operating frequency of the X5 crystal as the synthesizer in the SI5351 is quite flexible.

We can use one of the frequency selections permanently stored in the internal frequency table to load the SI5351 in order to measure the frequency offset error. Once we know how far off the crystal is operating, we can adjust the calculations (for the SI5351 register values) to configure the SI5351 for operation at that actual crystal frequency. Consult the AN1234, AN619 and AN551 application notes from *SkyWorks* for intimate details of calculating the values used to load the *Multisynth* registeers.

The approach taken when using the *SI5351 configuration table utility* (see section 17.1 on page 295) is to base the calculations on the target output frequency given the 20MHz crystal specified in the schematic. Although we could calculate backwards and arrive at the actual operating frequency of the crystal, that level of effort is not required for successful operation with a typical hand held VHF receiver.

We simply measure the uncorrected error offset and then apply that to the target output frequency. In practice this method limits the range of frequencies we can correctly generate (as errors accumulate away from the point at which we measured the offset). The typical hand held VHF receiver, however, will simply ignore small errors (i.e. the carrier can be several KHz off and the HT will still lock to the carrier).

The support in the zNEO operating software is a mechanism to access register values stored in FRAM (see section 9.15.2 on page 159) in the form of a record that looks somewhat like a command. The **FREQ** command can then scan the commands in FRAM looking for a matching entry to find the values that will be loaded into the SI5351 *Multisynth* registers.

Some additions were made to the V3.72 software as well as the *SI5351 configuration table utility* to make the frequency offset visible. The SI5351 doesn't make any use of this information (i.e. the offset frequency), we track it in the zNEO software as a documentation aid for the end user. The **STAT** command will display the frequency offset if it has been defined.

The *SI5351 configuration table utility* will generate an **FOFF** command that resides in the **INI=** file noting the offset frequency used to generate the external table. The **FREQ** command gives priority to the external frequency table. This means the the external frequency table can be used as the primary source of data for the *Multisynth* registers (in the SI5351). This also somewhat implies that the internal table need only have one entry to configure the SI5351 for frequency measurements. The internal table, in practice, needs this single entry for the table scanner in the frequency command to work correctly.

Some example external frequency table entries:

```
145.000=13BF,70E40,F4240
50.500=D26,61A7F,F4240,400
```

The frequency command scans for a matching string, requiring a character-for-character match. The three parameters in the entry are the patterns that will be loaded into the MSNA_P1, MSNA_P2, and MSNA_P3 registers in the first stage synthesizer in the SI5351.

The MSNB registers, although not used, are loaded with the same patterns as the MSNA registers.

The second stage *Multisynth* register values have the following default values (when not supplied as part of the frequency entry):

The MSx_P1 registers all have a value of 256 (0x100) loaded.
The MSx_P2 registers all have a value of zero (0x00) loaded.
The MSx_P3 registers all have a value of one (0x01) loaded.

For operation outside of the 2M band, the The MSx_P1 register pattern in the second stage *Multisynth* must also supplied.
Note that changing the second stage *Multisynth* value will change our operating band! Your band selection is seet by the low pass filter on the motherboard!

5.6.1 AN551 (Skyworks)

This application note indicates, in the first paragraph, that the SI5351 *utilizes a standard, non-pullable 25/27 MHz crystal*. This is not actually necessary, the SI5351 will work with other crystals. Do keep in mind that using the 20MHz crystal we shift things around that affect what range of frequencies we can actually generate. For operation in the 2M band, however, we see no issues.

The selected 20MHz crystal has a load capacitance specification the allows the crystal to operate above its specified frequency.

It is possible to select a crystal with a lower load capacitance specification.

5.6.2 AN619 (Skyworks)

This application note has the formulae needed to generate multi-synth register values.

As we are pulling the SI5351 reference oscillator (X5, CT1, CT2) around to achieve FM modulation, jitter is not particularly concerning to us (2.1.1. Selecting the Proper VCO Frequencies and Divide Ratios).

We only use (or enable) on CLK output at a time, so pin assignment doesn't present a concern (2.2. Output Clock Pin Assignment).

5.6.3 AN1234 (Skyworks)

Discussion of the crystal configuration in (2.1.1 XTAL Source).

Do **not** use the register map in the application note!

The register map in this document is for a 16-pin device and it the address assignments are incorrect for our MSOP-10 device!

5.6.4 Operating in the FM Broadcast band

The SI5351 can easily be programmed to operate in the FM broadcast band and the 4.07 software release allows selection of frequencies between 87.5MHz and 108.0MHz. An external frequency table entry is needed for the target frequency.

It is also necessary to carefully evaluate the output power produced by the transmitter. Moving down to 90MHz allows the RF amplifier modules to operate a bit more efficiently which will probably produce too much power on that band.

The 102-73161-22, 102-73161-24, 102-73161-29 and 102-73181-35 boards may be good candidates for use in this band.

The output filter network will need to be reworked to deal with 2nd. harmonics. The 102-73181-12 board has an updated table.

The standard rubber-ducky antenna will probably not work well as it will not present a proper 50Ω load to the transmitter and filter. You will probably need to use a telescoping antenna extended to about 71cM (28 inches). Several units listed on Amazon show a length of 30 inches.

5.7 Audio Filesystem Loading

With the addition of a voice feature to the 102-73161-25 and 102-73181 boards, we need a means of storing audio data for use with the **TALK** command. Given that the host interface (i.e. the command port) is intended to be operated by hand, we initially made no effort to implement a binary interface for loading audio waveform data. Rather, we simply adopted the use of an existing (and ancient) format from the early days of microprocessors. The command that effects the loading process is descriped in section 10.58 on page 215.

The *InTel HEX record* begins with a colon (:) which is recgonized by the command decoder in the **fox transmitter**. The *InTel HEX record* is decoded and loaded into the FLASH device. We try to make use of *page-write* FLASH devices to allow the FLASH device program time to overlap with the arrival of the next record. The *page-write* feature allows a line of data, usually at least 32 bytes, to be sent to the FLASH device to be written as a group. The write time for a line of data is the same as that required for a single byte of data.

We demand that the download data be no more that 32 bytes, in most cases exactly 32 bytes, and that is naturally aligned (that is is the 32 byte chunk start on a 32 byte boundary).

The *InTel HEX record* has an 8-bit length field, a 16 bit address field, an 8-bit record type field, up to 32 8-bit data fields, and an 8-bit checksum.

There are three additional *InTel HEX record* types that the command decoder deals with. There are two extended address fields that are used to provide upper address bits that are needed to address the typical FLASH devices we see on the **fox transmitter** board.

A type 2 record has a 16 bit address segment, that is shifted up 12 bits and added to the address in the data record.

A type 4 record has a 16 bit address extension that is shifted up 16 bits and added to the address in the data record.

And finally a type 1 record that is an EOF record that indicates the end of a load file. Although we recgonize and validate this record, it is not used.

A short delay is required between records to allow the command decoder to recgonize that a new command (or *InTel HEX record*) has arrived and copy the receive buffer to a local buffer before the next line of data can be processed by the **fox transmitter**.

As the command (i.e. the *InTel HEX record*) is being decoded and written to the FLASH device, the next line of the *InTel HEX record* can be moving into the input buffer.

The command decoder sends out a status report after the *InTel HEX record* has been fully processed taht may be used to implement a closed-loop loader. In practive, the **fox_simple** utility runs open-loop with a programmable delay between lines in an attemp to take some advantage of the processing overlap that is available.

You should note that there is no *file system* implemented in the FLASH device. Memory allocation is expected to be performed by the utility that is used to gather all the needed audio clips together for downloading into the **fox transmitter**.

As implemented, the audio utility (described in section 15 on page 273) is given a list of WAV files that will be used by the **fox transmitter**. The audio utility then takes the WAV files in sequence and generates *InTel HEX record*s is order. Each new WAV file starts on a convenient (at least 32 byte) boundary (to make debugging a bit more convenient) with the WAV file being copied, in effect, to the HEX file. Embedded in the HEX file after each audio clip are a directory record for the audio clip.

The directory records are prefaced with a **REM-** (see section 10.2.3 on page 170) to keep them from actually being saved as directory entries in the FRAM.

After the HEX image for the load file is generated we use **grep** to extract the directory entries from the HEX image. The **REM-** is removed using a text editor so that the directory entries may then be save to the FRAM.

5.7.1 Binary High-Speed Loading

The author has recently implemented a high-speed loader to speed up loading of the audio filesystem.

This is a closed-loop protocol using more-or-less standard ASCII characters to implement the protocol.

A terminating message is used to automatically switch back to normal operation.

We enhance the **56K** and **H115** commands to switch over to operating in the binary protocol mode at the specified bit rate.

This reduces audio waveform load time considerably. We gain from the increased transfer speed (when using **H115**), and by having the zNEO reply once the FLASH programming cycle has finished.

Additional discussion of the high-speed loader is found in section 12.4 on page 239.

5.8 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** running with 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.

You can monitor activity in a fully configuired device, as well.

5.8.1 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 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 few 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.

Assuming you have sufficient space in the FLASH, you can load any number of audio clips to form the fox transmitter personality.

5.8.2 Initialization

Once the audio waveform data is loaded into waveform memory (using an InTel HEX file) and the audio filesystem directory 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 an example of the **INI=** file that is run at startup.

```
esav INI=TIME
esav INI=WAIT 0.5
esav INI=TIME
esav INI=EPOC -5.0

esav INI=CALL KA0AAA
esav INI=NAME FOX20

esav INI=CONF DRA818
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. These setup commands should be at the begining of the **INI=** file to establish the transmitter identity. This allows the *<call>* and *<name>* substitutions to function later in the **ANN=**, **TEST=** and **MAS=** files.

Pay particular attention to the *callsign* and *nickname*. The *callsign* **must** be defined in the setup commands to allow the transmitter to remain in compliance with FCC rules. The *nickname* will be used to uniquely identify each transmitter as they will all be operating under the same callsign.

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 DRA818 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 DRA818/SA818 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 144.150MHz.

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.

5.8.3 Announce

The announce message is sent shortly after 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 E V 7.2
esav ANN=BATC E I
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 5.3.1 on page 99).

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 (i.e. the ANN= commands) message.

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

Once the code traffic clears we call up the callsign and nickname in waveform memory and set it out (verbally) to let the operator know he has planted the intended 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 one decimal place. 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.

5.8.4 Active Scheduling

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.

Assuming the **ANN=** file has enabled at least one schedule, the operating program, when idle, looks for active schedules that are to be run.

5.9 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 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.

5.9.1 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                                    1
esav INI=NAME FOX20                                      2
esav INI=TIME                                            3
esav INI=EPOC -6.0                                       4
esav INI=CWPM 25                                         5
esav INI=CONF SI5351                                     6
esav INI=CONF CLK0                                       7
esav INI=FREQ 144.150                                    8
esav INI=MODS S0 300 0                                   9
esav INI=MODS S1 300 60                                  10
esav INI=MODS S2 300 120                                 11
esav INI=MODS S3 300 180                                 12
esav INI=MODS S4 300 240                                 13
                                                         14
esav ANN=BEGN                                            15
esav ANN=BATV V                                          16
esav ANN=BATV I                                          17
esav ANN=BATV R                                          18
esav ANN=DONE                                            19
esav ANN=FREQ 144.450                                    20
esav ANN=RUN0,S2                                         21
esav ANN=STAT                                            22
```

The setup proceeds more-or-less normally through line 8, although we specify the startup frequency (in this example 145.150MHz). 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.

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 **RUN0** 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 22 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.

5.9.2 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 (nominally 144.150MHz) 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). On 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.

5.10 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 4.4 on page 65.

Mating housings for the various configurations:

Table 5.4: 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.

5.10.1 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..

5.10.2 FILT_TONE

Filtered VCMO_TONE.

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

5.10.3 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.

5.10.4 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.

5.10.5 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.

5.10.6 V9.0

Switched battery voltage.

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

5.10.7 SWITCH

Panel switch input.

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

5.10.8 PHOTO_CELL

Photo Cell input.

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

5.10.9 GNDP

Transmitter ground.

5.11 CHRP: Chirping

A special modes was added to the V3.50 software, a *"chirp"* mode. The base mode of operation 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 capability to emulate an audio chirp was added to the V3.84 software update. This is accomplished by adding an audio file capability to the CHRP command. The tone field is replaced with the name of an audio file that will be sent rather than a simple tone.

An external audio processing utility (in the authors case, *audacity*) is used to generate an audio chirp that is stored in FLASH.

The **period** and **duration** arguments function as expected, controlling the timing. The**rep** argument, i.e. the number of time the chirp loop repeats is identical, being controlled by the last argument.

The **duration** argument acts slightly different when sending an audio file. Since the audio file itself determines the length of time the transmitter will be active, the **duration** argument sets an idle time, after the transmitter is sending carrier, before audio is sent.

The command to initiate this operating mode is the **CHRP** command along with a few parameters that control its operation. See section 10.2.25 on page 196 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>**).

5.11.1 R68 in alternate position

This is the position that FOX21..FOX32 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.

Use the 102-73181-28 RF amplifier board when R68 is in this position. The 102-73181-28 board has an on-board power swith connected to the PTT net to perform this function. This allows the DRA818/SA818 to be used with the 102-73181-28 RF amplifier board without having to change R68.

V3.76 software is required to operate with tight schedules, that is where an overlapping group is sending chirps less that 5 seconds apart.

5.11.2 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 too short for proper operation.

5.11.3 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).

5.11.4 Developing a CHIRP Sequence

This is a brief discussion of how to make use of the **CHRP** command to setup a simulated wildlife tracker.

This attempts to emulate a VHF tracker while living within the constraints of part 97 (i.e. so we remain rules compliant as far as station identification is concerned.)

To operate as a **CHIRP** transmitter, we make use of only 3 commands. First, the **BEGN** command enables the RF subsystem and sends out station identification in code. We then use a **CHRP** command to generate a series of audible chirps. The sequenced duration of this command should be short enought that we can identify every 10 minutes, as required by the rules. Finally, the third command is the **DONE** command that send station I.D. and shuts down the RF subsystem.

As you should expect, in order to have this cycle repeat we need to have a schedule set up to trigger this sequence at regular intervals.

The easiest approach, perhaps, is to set up a short schedule and a long **CHIRP** sequence. Save (esav) something along the lines of the following sequence:

```
S5=CWPM 25
S5=TONE 1.0
S5=BEGN
S5=CHRP 1.2, 1, 0, 0.10, 560
S5=DONE SILENT
```

This sequence takes a bit more than 9½ minutes to run. The **BEGN** command and **DONE** command take almost 20 seconds to run at 25WPM. We explicitly force the code rate to 25 WPM to constrain the time the **BEGN** and **DONE** commands require to execute.

This schedule can be set to run every 60 seconds which will cause the entire chirp sequence to start on the minute. Any existing **ANN=RUN0,S0** command will need to be removed to avoid the **S0** schedule creeping in unexpectedly. Save (esav) these commands (remove any other **RUN0** commands):

```
INI=MODS S5,60,0
ANN=RUN0,S5
```

Now, when the transmitter is reset, the S5 sequence will be setup to run at the start of the minute and the schedule enables with the **RUN0** command.

Shortly after power on, the S5 sequence sequence will run, with a signon message, then 9½ minutes of the simulated chirp, followed by the signoff message. At 25WPM this will take a bit less than 590 seconds to run.

The transmitter will go silent for about 10 seconds and then start up again.

5.12 Modulation

The fox transmitter can be operated in A1A (keyed unmodulated carrier) mode or in F1A/F3E (FM modulated voice).

The 102-73181-10 boards have the power switching required to correctly implement this function.

The 102-73181-5 boards have the mechanical interface (mounting) required to securely attach the 102-73181-28 board.

5.12.1 A1A

Disable the tone generator by sending **TONE 0.0**.

Enable the *A1A* mode by sending **CONF CW** or **CONF AM**.

The transmitter will now send an unmodulated carrier for the dit/dah elements. The carrier is diabled between elements.

This will not be generally compatible with an FM radio unless the squelch is open. The use of a DTOA antenna system should work with this configuration.

Voice traffic (i.e. the **TALK** command) is handled somewhat gracefully. Each voice fragment (or file) enables and diables the carrier.

5.12.2 F1A/F3E

This is generally compatible with an FM radio. The carrier is enabled for the duration of the message.

Code traffic is handled by sending an audio tone over the air.

5.13 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.

5.13.1 Status Reports

- System status

  Software version number
  System Time
  Time Zone Offset

- TOY clock status

  Time and control bits from DS1672

- Update Flag

  Indicates that some setup instructions have run

- Jumpers (TEST and MAS)

  System view of the TEST and MAS jumpers

- External memory devices

  JEDEC ID data from the two external memory devices

- Battery status

  Analog readings from the battery (voltage and current) when Idle
  Analog readings from the battery (voltage and current) while Transmitting

- Analog channels

  Analog readings from the remaining analog channels

- UART buffer status

  UART input buffer use counts

- Schedules

  List of the loaded schedules

- Identity (callsign, nickname)

  Callsign (**CALL** command)
  Nickname (**NAME** command)

- zNEO port status

  Current state of the zNEO-GPIO port bits

- Radio Configuration

  Short summary of the currently loaded radio configuration
  Contents vary with selected RF generator.

- CW Configuration

  Morse Generator configuration

- Transmission State Delays

  Accommodation delays (in mSec)

5.13.2 Configuration Reports

- RFGEN

  Radio Personality selection flags

- RADIO

  Transmitter control flagss

- CWISR

  Tone Generator enable flag

- AUDIO

  Voice generator configuration flags

- SYNTH

  102-73161-25 control flag

- ANALOG

  System analog channel monitor flags

- DEBUG

  Debug flags

- SYNTH

  SI5351 Synthesizer control flags

- BMON

  Battery voltage monitor coefficients flag

- VOICE

  Voice file system location flag

- T<n> timing

  Transmission State Delays

Chapter 6

Assembly

Assembly Hints

These notes are general suggestions about the order of installation and issues encountered by the author building multiple units.

The assumption is that you are a competent technician. This is, in no way, a hand-holding tutorial!

6.1 Board Procurement

The author makes use of JLCPCB (URL: https://jlcpcb.com/) to fabricate circuit boards. This vendor has attractive pricing and a convenient web interface for ordering boards.

There exists a ZIP file (102_73181_10.jlcpcb.zip) that contains all the files JLCPCB requires. The files match the JLCPCB file naming convention allowing the web interface to correctly identify copper, silkscreen and soldermask layers.

The PWA drawing (102_73181_10_pwa.pdf) has assembly notes. Only the top and bottom layers appear in this drawing.

All of the mechanical parts required for the finished project appear on this drawing. This serves to place the mechanical parts into the *Bill of Materials* so you have an idea of the screws and spacers required when mounting the main board into the enclosure and the daughter-board to the the main board.

Refer to the PWB drawing (102_73181_10_pwb.pdf) for board dimensions. This drawing is provided to the board vendor and lists all the board fabrication requirements. All the individual layers are broken out in this drawing. The layering order on the first page of this drawing indicates the number of layers required.

6.2 Board Inspection

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

6.3 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 (those with values expressed as *UF*).

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.

6.4 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.

6.5 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 *Comp S/N* column indicates the side on which the part is located as well as the parts schematic location.

The *Idx#* column number matches up in all the parts lists. Later board layouts, i.e. the *102_73181_10_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.

Most tantalum caps and inductors can be installed next. Delay installing large parts that may impede access to active parts (i.e. the X4 crystal gets in the way of installing U22).

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 and the 5V regulator. See the regulator notes in the next section. The backup battery will be installed after some initial testing.

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

6.5.1 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°.

Also verify that the regulator will provide adequate current if you plan to make use of the DRA818/SA818 modules. If you avoid the DRA818/SA818 modules altogether, a 500mA regulator is sufficient.

6.6 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 allows the RF daughterboard to fit within the specified enclosure. It also provides adequate room for the BNC connector.

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 and the associated motherboard connectors must be mated and then soldered (do **not** solder these connectors unless they are mated with the boards fastened together with the 12mm spacers). 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 PBC36SAAN or PBC36SABN) with a mating length of 0.230" and a tail length of 0.120" or 0.230". 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. We assemble these with the RF daughterboard mated to a motherboard to get the vertical spacing right and to force proper alignment of the connectors.

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.

Chapter 7

Haywires

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

Currently, the 102-73181-10 board are built **without** any haywires.

7.1 Audio/Voice

This wiring hack is available to enable audio capability on the 102-73161-25 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 (680Ω or 820Ω will also work) resistor and suitable sleeving.

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 90º 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.

7.2 TOY Clock Battery Maintenance

This affects the 102-73161-25 and 102-73181-0 boards.

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.

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.

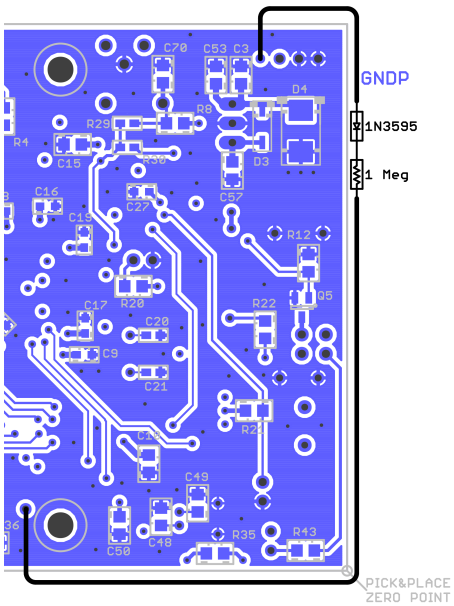


Figure 7.1: Charge Circuit

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.

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.

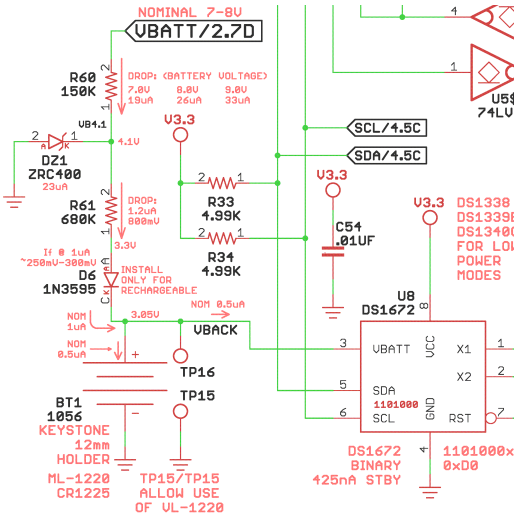


Figure 7.2: Regulated Charge Circuit

Chapter 8

Commissioning

Initial Testing

8.1 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 (left edge of board).

Check positive end of C4 for battery voltage.

Check 5V regulator at vias below the 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.
The programming cable may prevent the zNEO from executing instructions, you may find it necessary to remove it for the zNEO to operate.

Attach a USB serial cable to J4 and reset the system looking for a prompt through the serial/USB connection. Communication settings are 57,600 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 8.1: Startup

```
sts01,00* ************************************************************
sts01,01* KC0JFQ  FOX Transmitter  V3.71
sts01,02*   Z16F2810AG20EG
sts01,03* Tools Ver: 20230510  <ROM:26374 EROM:87560 Flash:113934>
sts01,04* Flash Prog 04.26.03 MB85RS512T    FLASH_FRAM       Fujitsu   512K-bits   2048-records CMD3
sts01,05* Flash WAVE C2.20.19  MX25L256  FLASH_PAGE32      Macronix 262144K-bits   4000-seconds CMD5
sts01,06* ************************************************************
```

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

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

Failed Software Load

Arrgh! the ZNEO seems to have been damaged?

The author has encountered a situation where the oscillator circuit in the ZNEO is misconfigured and will not function. Injecting a 10MHz to 20MHz clock into the ZNEO on the **XIN** pin (C21 on the bottom of the board) may allow you to program the device.

Once the device has been successfully programmed, the *Flash Option Bits* should now be programmed to correctly configure the oscillator.

8.2 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.

8.2.1 Xtal Test, SI5351 (20MHz)

The SI5351 can be configured with an RF daughterboard that is specifically made for development testing. This board 102-73181-60, provides an SMA connector that is connected to the **CLK0** output of the SI5351.

When fitted to the transmitter mainboard, the SMA connector J4 is connected to a frequency counter that need only deal with the 20MHz crystal frequency (rather than the 144MHz carrier).

If you are making use of the trim caps on the SI5351, command the device to route the SI5351 oscillator net directly to the **CLK0** output pin using an SI5351 diagnostic command.

Connect J4 to the frequency counter.

Issue the **5351 XTALT** diagnostic command.

Trim CT1 and CT2 to obtain 20MHz.

Once you have obtained the correct frequency, power down, remove the test jig, and reinstall the RF daughterboard.

8.2.2 Xtal Test, SI5351 (144.100MHz)

The SI5351 crystal offset can also be measured if you can measure the carrier frequency (i.e. up at around 144MHz). A **tiny SA** analyzer may be used if it is calibrated.

This procedure is performed prior to loading the FRAM.

If using a frequency counter, the SMA connector (J4) on the test board (102-73181-60) may be used. If using the **tiny SA** mount any of the low power RF amplifiers the motherboard and attach a 50Ω load (or the antenna) to the output BNC. The **tiny SA** will see adequate signal with its telescoping antenna attached. Start with the **tiny SA** antenna collapsed!

Command the frequency to 144.100MHz:

FREQ 144.100

Enable the carrier.

BEGN SILENT

Use the test instrument to measure the actual operating frequency.

Disable the carrier.

DONE SILENT

Calculate the required offset and select the corresponding frequency setup table.

Use the *fox_simple* utility to load the selected frequency table.

Now that the frequency corrected table is loaded into FRAM, test by selecting a few frequencies:

FREQ 144.150

BEGN SILENT

Measure the carrier frequency.

DONE SILENT

FREQ 144.325

BEGN SILENT

Measure the carrier frequency.

DONE SILENT

The two frequencies should be reasonably close.

8.3 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 in place on the 102-73181-36 daughterboard.

Configure for the DRA818.

Select the operating frequency (i.e. FREQ 144.150).

Send the **BEGN SILENT** command to send RF.

Perform RF tests (i.e. verify frequency is correct)

Send the **DONE SILENT** command to stop RF.

8.4 Install backup battery

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

You may check to see that voltage is present at the battery when the main (6-cell pack) battery is installed. You should see around 3 volts at the positive backup battery pad from the main battery (6-cell pack). Remove the main battery before installing the backup battery.

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

Use a terminal program to configure the DS1672 for operation using **TOYC NONE**. You may then use the schedule loader to set the TOY clock to the current time.

8.5 Loading FRAM and FLASH

Use the host tools to load the operating configuration commands and waveform data.

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

Keep in mind that the *ERAS DEV* command clears all of FRAM and will require reloading the waveform data if it is stored in FRAM (102-73161 artwork). 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 support 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.

8.5.1 Required Audio Files

For battery condition reports there need to be a set of voice utterances present in the audio file system. In addition to the battery condition fragments we need the callsign and nickname utterances to be present.

You should, of course, perform a complete checkout on the bench prior to any deployment to verify the voice clips can be heard and understood.

In any event, the Fox Transmitter will emit a callsign in Morse code at the beginning and end of message traffic. That behavior is built into the basic operating software.

Battery Condition Reporting

To verbally report on battery condition the audio clips listed in table 8.1 must be present in the audio file system. Their position within the file system is unimportant, they simply must be present.

These clips were all recorded for the prototype units and may be used unchanged if you so desire. The voice isn't all that great to listen to, so you may prefer to create your own clips.

Table 8.1: Battery Condition Voice Clips

Audio Filename	**Utterance**
BATTV	"battery voltage"
BATTI	"battery current"
REG5	"5 volt regulator"
V_VOLT	"volts"
V_MAMP	"milli amps"
POINT	"point"
V_N0	"zero"
V_N1	"one"
V_N2	"two"
V_N3	"three"
V_N4	"four"
V_N5	"five"
V_N6	"six"
V_N7	"seven"
V_N8	"eight"
V_N9	"nine"

Operating Frequency Reporting

If you have worked through the **ANN=** sequence found in section 19.1.6 on page 324 you will note that there are a couple of **TALK** commands with a *'freqM'* and *'freqK'* substitution.

These **TALK** commands announce the operating frequency during the announce message and will require audio clips be present for this report to be voiced.

This feature is implemented in the **ANN=** sequence and the *fox_simple* utility. It does not depend or use anything in the zNEO code other than the **TALK** command.

The table lists the frequency clips loaded into the authors units. If there are other frequencies you intend to operate on, simply capture the utterance and add it to the audio file system.

Table 8.2: Frequency Announce Clips

Audio Filename	Utterance
V_F144	"one four four dot"
V_F145	"one four five dot"
V_F200	"two zero zero"
V_F225	"two two five"
V_F250	"two five zero"
V_F275	"two seven five"
V_F300	"three zero zero"
V_F325	"three two five"
V_F350	"three five zero"
V_F375	"three seven five"

Callsign and Nickname Reporting

You will, in any case, need to digitize utterances for your station callsign and all the nicknames you require in order to operate.

This example is the audio clips used by the *Iowa City Amateur Radio Club*.

Table 8.3: Station Announce Clips

Audio Filename	Utterance
KC0JFQ	"Kay Cee Zero Jay Eff Queue"
W0JV	"DoubleU Zero Jay Vee"
FOX20	"fox twenty"
FOX21	"fox twenty one"
FOX22	"fox twenty two"
FOX23	"fox twenty three"
FOX24	"fox twenty four"
FOX25	"fox twenty five"
FOX26	"fox twenty six"
FOX27	"fox twenty seven"
FOX28	"fox twenty eight"
FOX29	"fox twenty nine"
FOX30	"fox thirty"
FOX31	"fox thirty one"
FOX32	"fox thirty two"
FOX33	"fox thirty three"
FOX34	"fox thirty four"
FOX35	"fox thirty five"
FOX36	"fox thirty six"
FOX37	"fox thirty seven"
FOX38	"fox thirty eight"
FOX39	"fox thirty nine"

Loading all these nicknames requires more time when loading the FLASH memory. It saves considerable headache, however, when generating operating sequences for eighteen or twenty transmitters.

The individual utterances, the audio files, all in the same location in all the FLASH devices. This allows using a single copy of the **TALK Directory**.

With the **TALK Directory** located in FRAM, erasing and reloading FRAM becomes far easier.

Local Audio

You are, of course, free to add to the audio in the FLASH. Given a larger device, you can store considerable period of speech or sound effects.

As many of the speech clips were developed by the author, the audio *files* were added a bit at a time to the audio file system. The FLASH device is erased to 0xFF pattern and you may change bits to a zero at any time. You are not, in any way, required to load the FLASH in one go.

During development. you can record updates in the free space of the FLASH device (i.e. at the end of the current audio data) and the simply update the **TALK Directory** as needed.

Table 8.4: Silly Voice Clips

Audio Filename	Utterance
FD_FOX	"I am your field day fox transmitter"
FD_GAZELLE	"Hey look at me, I am now a Gazelle"
FD_CATCH	"Hey look at me, Catch me if you can!"
FD_TUNA	"I am a... Tuna Fish sandwich"
FD_SILLY_8K	"All right now, this is getting just plain silly"

This shows a few of the audio clips loaded into the ICARC fox transmitters. Although these clips reside in the FLASH device, it is not necessary to actually use them. You are free to leave test clips in place or have special purpose clips stored in the audio file system.

8.6 Power Evaluation

Once a working sequence has been loaded, you can evaluate the performance of the battery you will be using. This requires exhausting a set of batteries, giving you an idea of a good setpoint for the battery reporting command **BATC**.

Start by connecting the antenna connector to a dummy load so you are not transmitting constantly as this evaluation may take over a day to complete. Remove the **TEST** and **MAS** jumpers so you will operate normally.

Verify that the sequence has **BATR** commands when idle and active: see section 19.5 on page 334.

Connect to the target Fox Transmitter with a serial cable and log the traffic using a logging program that time-tags each line of output (such as the *halo_term* utility).

Switch the Fox Transmitter on and verify that the logging program is collecting data with timetags. Now simply let the transmitter run until the battery runs out and it stops.

Once the Fox Transmitter has stopped operating, the data in the log file may be extracted and plotted (for an example see section 4.10.2 on page 61). You may notice that the 6-cell pack needs service at this point as it will not have enough power left to power the RF section.

The regulated 5V channel may be used to determine when the 5 volt regulator falls out of regulation. At this point the RF subsystem will produce lower power as it runs from the 5V rail.

At some point, the 5V channel will fall out of regulation, the 5V channel showing below 5 volts. Move back on the plot in time several hours to pick the trip voltage where the Fox Transmitter will add an **SOS** call to the battery report (from **BATC** command). You can listen for this when you prepare the Fox Transmitter the night before a hunt.

If you select the battery voltage trip point (in the **BATC** command) correctly, you will have enough battery to run an entire hunt once the battery low report (i.e. the **SOS** pattern in the **BATC** report).

Consider you will have to find your transmitters at the end of the hunt. Having and active transmitter to track may prevent a lost unit!

Take note of the plot in section 4.10.2 on page 61. The current ramps up as the battery terminal voltage falls. We eventually reach a point where internal battery resistance prevents supplying enough current to maintain the 5 volt output from the regulator.

The 3.3V regulator for the digital logic remains in regulation allowing the zNEO to continue to provide battery reports for a short period after there is no longer a stable 5V rail.

The example plot shows a Fox Transmitter with a low power RF amplifier (100mW or less). Using higher power RF amplifiers will increase current requirements when the RF amplifier is active drawing the battery down faster. This, of course, will move the point where the batteries can no longer supply enough current to the regulator to keep the output in regulation.

This example would indicate that a cutoff voltage of 7.20 volts (i.e. set the battery report to **BATC VE 7.20**) would be appropriate. This will begin sending the battery low indication (i.e. code message **BATC SOS TTTTTT EEEEEEEEE**) when there is about 10 hours of operation left for the unit.

Do keep in mind, however, that using an alternate power amplifier affects the trip voltage that should be employed.

Chapter 9

Software

Breakdown of the zNEO software subsystems.

9.1 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.

9.1.1 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.

9.1.2 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 (hence the DS1672 TOY clock).

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 more than adequate.

The zNEO makes use of an internally derived 10mS interrupt to update the system time. The TOY clock is read at startup (using the **TIME** command) and stored in the system time field. The system time advances 10mS at every 10mS interrupt.

Note, at this point, that the TOY clock doesn't track sub-seconds. Prior to V3.76 software, we may choose to load the clock in a synchronous manner (which we do), we spent no effort in the fox transmitter system to synchronize at the sub-second level. This meant that the group of fox transmitters would be operating up to 1 second apart.

The V3.76 update changes the behavior of the **TIME** command to get the system time closer to that held in the TOY clock. The **TIME** command polls the TOY clock waiting for the seconds to change (the polling activity will time-out after one second, so hardware faults will not hang things up).

When the seconds field in the TOY clock changes, the system time update finally occurs (which will zero the sub-seconds field). The fox transmitter system time will now be within the RTI granularity (10mS) of the TOY clock.

Assuming the TOY clock was set synchronously (which the *fox_simple* utility does) all of the fox transmitters should be set within 10Ms of each other.

9.2 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!).

9.2.1 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.

9.2.2 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.

9.2.3 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, the night before...).

Note that the TOY clock may need to be read twice at startup (see section 4.8.2 on page 52).

9.3 Scheduling Flexibility

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

Table 9.1: 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 9.2: 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.

9.4 Parameter Substitution in the Fox Transmitter

There is a simple parameter substitution machanism in the *Fox Transmitter* itself. The goal is to move all unit-specific setup into the **INI=** file while keeping all unit-specific information out of the operating sequences.

The aim is to allow the sequence to be shared, unchanged, between all units in the group.

9.5 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. Notes on successfully setting the time are found in section 4.8.2 on page 52.

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.

9.5.1 Clock Characteristics

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

The time must loaded into the TOY clock prior to first use. The DS1672 has a backup battery that maintains the time once loaded 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 battery is present.

9.6 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 processing and transmission. Long messages may be built up of any number of short buffers that will fit in the FRAM.
The message text is translated into many code chips that are used to drive the interrupt routine controlling the tone generator.

9.6.1 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*.

9.6.2 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.

9.6.3 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 10.2.25 on page 196

9.6.4 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 9.1: cmd_code.h-59

```
#define CODE_DIT                '<'                                     59
#define CODE_DAH                '>'                                     60
```

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

Listing 9.2: cmd_code.c-123

```
//        Total = 50 elements                                           123
//        () = intercharacter                                           124
//        [] = interword                                                125
//                                                                      126
//      chipping rate in seconds is 1.2/WPM                             127
//      multiply by the reference clock into the                        128
//      timer to set the interrupt rate...                              129
//                                                                      130
//                                                                      131
                                                                        132
    rom struct CODE_TABLE code_table[] = {                              133
                    {' ',  R"W"        },   // inter-word spacing      134
                    {'_',  R"W"        },   // inter-word spacing      135
                    {',',  R"W"        },   // inter-word spacing      136
```

Timing characters.

Listing 9.3: cmd_code.c-138

```
{':',  R"S"       },   // colon (in time string)
{CODE_DIT, R"."   },    // dit
{CODE_DAH, R"-"   },    // dah

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

Punctuation characters.

Listing 9.4: cmd_code.c-154

```
{')',  R"-.--.- " },   //
{'\'', R".----. " },   //
{'_',  R"..--.- " },   //

{'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"...- "    },
```

Alpha characters.

Listing 9.5: cmd_code.c-181

```
                    {'X', R"-..-  "    },
                    {'Y', R"-.--  "    },
                    {'Z', R"--..  "    },

                    {'0', R"-----  "   },
                    {'1', R".----  "   },
                    {'2', R"..---  "   },
                    {'3', R"...--  "   },
                    {'4', R"....-  "   },
                    {'5', R".....  "   },
```

Number characters.

Listing 9.6: cmd_code.h-16

```
//
//        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.

9.6.5 Interrupt Activity

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.

The interrupt service routine services each interrupt by decrementing the CHIP_COUNT field to zero and moving on to the next chip.

At the start of a new chip we first look at the CHIP_ACTIVE bit, looking for an end-of-buffer indicater (when chip is zero). The interrupt routine then deals with the CHIP_KEY_ON bit, which is the state for the TONE_ENABLE net.

The CHIP_GAPPED is used for debugging.

9.6.6 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.

9.7 Audio

Audio waveform 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.

9.7.1 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*>,<*start*>,<*count*>,<*address*>

TALK=<*name*>,<*start*>

9.7.2 Waveform Data

Loaded into FLASH 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, 8K/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 or 16 bit data.

9.8 Status Reports (commanding)

Whenever an activity occurs within the system, a status report is produced and delivered to 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.

9.8.1 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.

9.8.2 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.

9.8.3 sts

This report (note that *sts* keyword is lowercase) is part of any intermediate result reporting.

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 (begining reports from a **HELP** command):

```
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.

The subsequent text is generally human readable.

Reports that are intended for computer consumption are typically formatted as *KEY=Value* pairs. See the sample report in section 4.10.2 somewhat after page 61.

9.9 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  V3.71
sts01,02*   Z16F2810AG20EG
sts01,03* Tools Ver: 20230510  <ROM:26374 EROM:87560 Flash:113934>
sts01,04* Flash Prog 04.26.03 MB85RS512T    FLASH_FRAM       Fujitsu   512K-bits   2048-records CMD3
sts01,05* Flash WAVE C2.20.19  MX25L256  FLASH_PAGE32      Macronix 262144K-bits   4000-seconds CMD5
sts01,06* ************************************************************
```

9.9.1 sts01,01: Version

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

9.9.2 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.

9.9.3 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).

9.9.4 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.

9.9.5 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.

This report, taken from FOX29, shows a very large FLASH device is installed on the board. The *MX25L256* is large enough to require sending it a 32 bit address (indicated by the FLASH_PAGE32 string).

9.10 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 31 2025 15:50:10 V3.89 fox_73181.c*
sts09,02* IDENT:Mar 31 2025 15:50:28 V3.05 gpio_local.c*
sts09,03* IDENT:Mar 31 2025 15:50:27 V2.05 timer_local.c*
sts09,04* IDENT:Mar 31 2025 15:50:28 V1.03 uart_local.c*
sts09,05* IDENT:Mar 31 2025 15:50:30 V3.12 flash_local.c*
sts09,06* IDENT:Mar 31 2025 15:50:29 V1.03 i2c_local.c*
sts09,07* IDENT:Mar 31 2025 15:50:29 V1.00 analog_local.c*
sts09,08* IDENT:Mar 31 2025 15:50:11 V3.23 command.c*
sts09,09* IDENT:Mar 31 2025 15:50:13 V1.03 cmd_help.c*
sts09,10* IDENT:Mar 31 2025 15:50:12 V1.04 cmd_code.c*
sts09,11* IDENT:Mar 31 2025 15:50:13 V3.27 cmd_stat.c*
sts09,12* IDENT:Mar 31 2025 15:55:10 V1.19 cmd_conf.c*
sts09,13* IDENT:Mar 31 2025 15:50:16 V1.16 cmd_message.c*
sts09,14* IDENT:Mar 31 2025 15:50:17 V3.10 cmd_sa818b.c*
sts09,15* IDENT:Mar 31 2025 15:50:17 V1.10 cmd_si5351.c*
sts09,16* IDENT:Mar 31 2025 15:50:20 V0.02 cmd_ics525_dummy.c*
sts09,17* IDENT:Mar 31 2025 15:50:20 V1.08 cmd_sys.c*
sts09,18* IDENT:Mar 31 2025 15:50:20 V1.00 cmd_proc.c*
sts09,19* IDENT:Mar 31 2025 15:50:21 V1.01 cmd_sched.c*
sts09,20* IDENT:Mar 31 2025 15:50:21 V1.05 cmd_time.c*
sts09,21* IDENT:Mar 31 2025 15:50:21 V3.04 cmd_test.c*
sts09,22* IDENT:Mar 31 2025 15:50:22 V3.14 cmd_voice.c*
sts09,23* IDENT:Mar 31 2025 15:50:23 V1.12 cmd_battery.c*
sts09,24* IDENT:Mar 31 2025 15:50:22 V3.05 cmd_flash.c*
sts09,25* IDENT:Mar 31 2025 15:50:23 V3.08 cmd_frequency.c*
sts09,26* IDENT:Mar 31 2025 15:50:23 V1.02 cmd_wav.c*
sts09,27* IDENT:Mar 31 2025 15:50:27 V1.06 fox_schedule.c*
sts09,28* IDENT:Mar 31 2025 15:50:12 V2.01 radio_control.c*
sts09,29* IDENT:Mar 31 2025 15:50:30 V1.00 daytime.c*
sts09,30* IDENT:Mar 31 2025 15:50:11 V1.00 modulus.c*
sts09,31* IDENT:Mar 31 2025 15:50:31 V3.22 intel_hex.c*
sts09,32* IDENT:Mar 31 2025 15:50:24 V1.00 test_help.c*
sts09,33* IDENT:Mar 31 2025 15:50:24 V1.00 test_spi.c*
sts09,34* IDENT:Mar 31 2025 15:50:24 V1.00 test_zneo.c*
sts09,35* IDENT:Mar 31 2025 15:50:25 V1.00 test_i2c.c*
sts09,36* IDENT:Mar 31 2025 15:50:25 V1.02 test_gpio.c*
sts09,37* IDENT:Mar 31 2025 15:50:25 V1.00 test_batt.c*
sts09,38* IDENT:Mar 31 2025 15:50:26 V1.01 test_code.c*
sts09,39* IDENT:Mar 31 2025 15:50:26 V1.01 test_serial.c*
sts09,40* IDENT:Mar 31 2025 15:50:26 V1.00 test_config.c*
sts09,41* IDENT:Mar 31 2025 15:50:12 V1.00 ident_scan.c*
sts09,42* KC0JFQ  FOX Transmitter  V3.95
sts09,43* Software Bld:   Mar 31 2025  15:50:13
```

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

9.11 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  V3.71
sts09,02* Software Bld:   Jun 24 2024  21:14:15
sts09,03* System Time:    16:49:23.740 (78563)
sts09,04* Epoch Offset:   19:00:00     (68400)
sts09,05* TOY Clock:      00ADE764 00 00; Osc ON, Charge Disabled
sts09,06* Sys Upd Flg:    Si5351_INIT
sts09,07* Conf Jumpers:   NOT_Master NOT_Test
sts09,08* Flash Prog  U3: 04.26.03 MB85RS512T    FLASH_FRAM       Fujitsu   512K-bits   2048-records CMD3
sts09,09* Flash WAVE U12: C2.20.19  MX25L256  FLASH_PAGE32      Macronix 262144K-bits   4000-seconds CMD5
sts09,10* Flash HEX Dev:  WAVE/FLASH/U12
sts09,11* Battery, Idle:  8.698V(0x037C)[9.751e-03]  40mA(0x0051)
sts09,12* Battery, TX:    8.483V(0x0366)[9.751e-03]  5mA(0x000A)
sts09,13* Analog Others:  Reg-5V: 5.110V(0x020C)[9.751e-03]  Switch: 0.000V  CdS-Cell: 0.000V
sts09,14* UART buffer:    143 (NET:0, USB:85)
sts09,15* <<<--- Scheduling  PARAMETERS --->>>
sts09,16* MOD Schedule    00 Idle    S0= 360 180
sts09,17* MOD Schedule    01 Idle    S1=  30   0
sts09,18* MOD Schedule    02 Idle    S5=  60   0
sts09,19* MOD Schedule    03 Idle    S6=  10   0
sts09,20* MOD Schedule    04 Idle    S7= 300   0
sts09,21* MOD Schedule    05 Idle    S9= 300   0
sts09,22* <<<--- TRANSMITTER PARAMETERS --->>>
sts09,23* Callsign:       W0JV
sts09,24* Nickname:       FOX29
sts09,25* zNEO Port Bits: OUT:E0,01,05,20,00,00,00,00 IN:F0,01,75,14,00,80,00,04
sts09,26* Radio Config:   0x1F0C43 SI5351 State-T0 TX_ENA TONE PWMH0 5MON SWIT PHOTO IMON VMON
sts09,27* Si5351 Config:  0x00B9 CLK0 8MA 8PF Offset:-10.000KHz
sts09,28* Si5351 Divisor: 0x13A5,0xA9EBF,0xF4240 0x0100,0x00,0x01
sts09,29* Frequency:      144.325 (Xtal: 20.000 MHz)
sts09,30* CW config:      30 WPM 1,3,7,14 [0x186A] 1.000KHz
sts09,31* State Delays:   T0:10 T1:50 T2:150 T4:50 T5:10
STS09,32* Handler_STAT I (cmd_stat.c*)  0.40 Sec
RDY00,00* (Sp=0xBF94)+1859 16:49:24.120
```

9.11.1 Software Bld:

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

9.11.2 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).

9.11.3 Epoch Offset:

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

9.11.4 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.

9.11.5 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.

9.11.6 Conf Jumpers:

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

9.11.7 FRAM 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.

9.11.8 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.

9.11.9 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.

9.11.10 Battery, Idle:

Voltage and current readings when the system is idle.

The 102-73161 boards will not report current.

9.11.11 Battery, TX:

Voltage and current readings when the system is transmitting.

The 102-73161 boards will not report current.

Notice that this report shows an unusually low current reading when transmitting. The high-RF environment affects Q2 resulting in an inaccurate current reading. More accurate readings may be obtained by loading the antenna port with 50 ohms when making current measurements.

9.11.12 Analog Others:

This reports the other analog channels that are present on the motherboard.

9.11.13 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.

9.11.14 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.

9.11.15 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.

9.11.16 zNEO Port Bits:

Diagnostic dump of the zNEO general IO port bit patterns.

9.11.17 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.

9.11.18 Frequency:

Currently selected transmit frequency.

9.11.19 CW config:

Code chipping parameters.

The word rate and timing parameters for the code generator.

9.11.20 State Delays:

To accommodate the various RF generator configurations, **BEGN** and **DONE** commands implement the delays defined in figure 4.15 on page 42.

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.

9.12 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 the 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).

9.12.1 I2C START

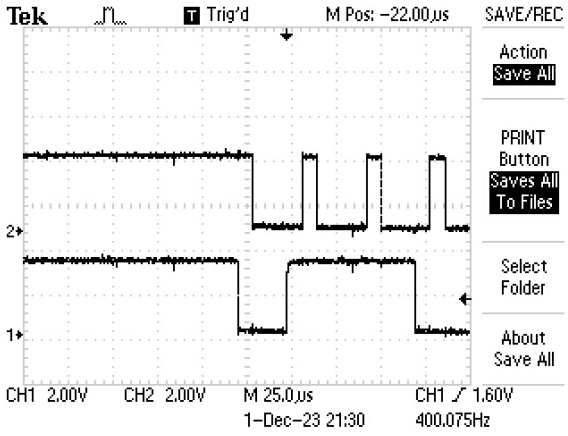


Figure 9.1: I2C START

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.

9.12.2 I2C Data

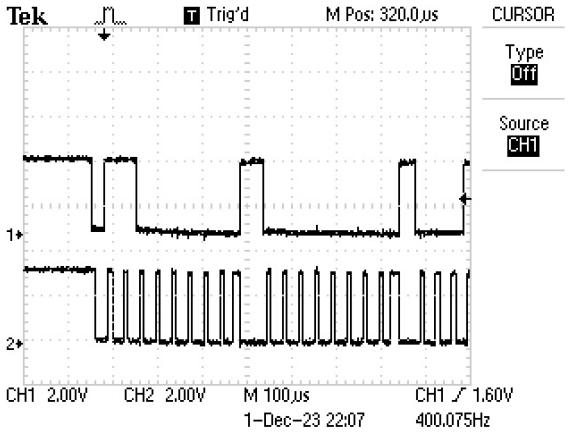


Figure 9.2: I2C Data

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.

9.12.3 I2C STOP

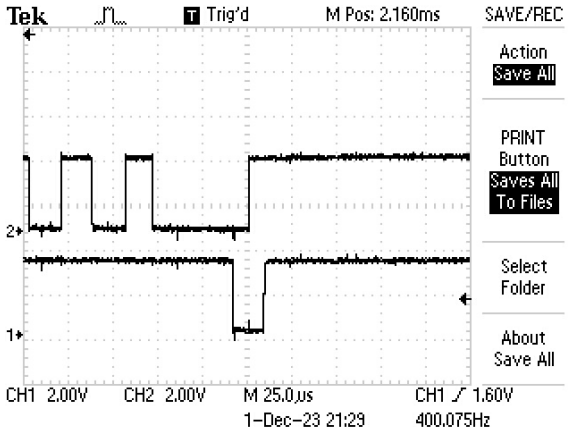


Figure 9.3: I2C STOP

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.

9.12.4 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.

9.12.5 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.

9.13 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 times 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.

9.14 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.

9.15 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 is used directly to setup the transceiver module.

A second method is to store the register setup patterns in FRAM using the **ESAV=** command. The frequency entries in FRAM are search before looking through the internal tables. This allows updated (i.e. corrected) frequencies to be loaded without having to reprogram the zNEO.

A third method is to specify the register patterns directly. This is primarily intended as a debug aid, but it can be employed when necessary.

9.15.1 Internal Frequency Tables

A small frequency setup table is present in the zNEO program image. To correctly generate the table the **actual operating frequency** of the reference crystal must be known. The load on the crystal affects its operating point so trim parts are supplied (CT1 and CT2). These can be installed although they tend to be a bit expensive. Another method is to generate the tables from the actual operating frequency of the X5 crystal.

The method used with the ICARC units was to program the zNEO with a nominal frequency setup table (generated assuming the crystal is operating at 20.000 MHz with a default offset of **0**KHz). The output frequency is then measured (with no modulation, in other words no audio) and an external table is generated with the actual offset required to correct for the transmitter to operate at the desired frequency.

In practice, it seems reloading the processor is faster than trying to select C58/C41 or trim out CT1 and CT2. The software suite is compiled with a set of frequency tables with offsets ranging up to 20KHz (i.e. si5351_frequency_table_0.c to si5351_frequency_table_20.c).

After flashing the fox transmitter with a baseline frequency setup table install the **TEST** jumper and run the **BEGN** command. After the signon message the fox transmitter is left with the RF generator on, the RF amplifier on, and no modulation on either the **VCMO_TONE** net or the **PWMH0** nets.

You can measure the carrier frequency and the select the appropriate offset from the group of si5351_frequency_table_*.c tables, recompile, reload, and then verify that the carrier is now where it needs to be.

See section on page 188 for a method of connecting the crystal to the **OUT0** pin of the SI5351 to allow meauring the crystal directly. This method allows using a less expensive frequency counter as the crystal is 20MHz, far below the carrier frequency.

We need only be within a half KHz for operating with an FM receiver, so don't expend enormous efforts trying to get is spot-on.

9.15.2 Frequency Tables in FRAM

Frequency setup tables can also be stored in sequence 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 or frequency counter available to find the carrier frequency, the SI5351 frequency calculation utility can generate a frequency table that is 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_5351-14.fox "
REM- Output File: freq_5351-14.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 in FRAM. This would be a set of frequencies that are not present in tables. The new register values are then loaded into FRAM.

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.

Given that the FRAM is searched before scanning the tables in program flash, we can totally override the internal tables if needed. We will, of course, need to have enough space in the FRAM to hold everything, so the size of the FRAM device may becomes an isssue.

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.

9.15.3 Direct Register Load

See comments on page 188.

Chapter 10

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*).

10.1 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.

10.1.1 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.

10.1.2 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.

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.

10.1.3 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 both set to ZERO.

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.

10.2 Command List

Table 10.1: 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)dent 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 10.2: 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 10.3: 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)
FOFF	PGM	offset	Frequency Offset (in KHz)
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 off dur cnt	Send chirp train
DONE	PGM	(SILENT)	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
BATR	PGM	flag	Power Evaluation Battery Report "I" battery coefficients

Sequence Commands.

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

Table 10.4: 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 10.5: 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 10.6: 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 MT**
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 10.7: 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	*b*	change bit rate to 57,600 b/S (V3.68-,V4.0+)
H115	FLASH	*b*	change bit rate to 115,S00 b/S (V3.69+)
:hex	FLASH-HEX	:llaaaattddddddddcc	InTel HEX loader (WAVE device)

FLASH Control Commands.

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

Table 10.8: Command List 8

Mnemonic	Class	args	Description
BATR	TEST		Battery state text report
TEST	TEST		Hardware Test Subsystem
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.

10.2.1 HELP

Table 10.9: Help Subsystem

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.

10.2.2 ONCE

Table 10.10: 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 (i.e. ONCE S0= *or* ONCE S1=).

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 it is measured from when the *ONCE* command is recognized to where control is returned to the *ONCE* command manager.

10.2.3 REM-

Table 10.11: Remark

Command sample	Description
REM-	Remark (no operation)
INI=REM-	Remark (no operation)
S0=REM-	Remark (no operation)

Action:

This is not acted upon.

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

Keep in mind that the file system, such as it is, uses the text to the left of the equal sign as a filename. Scaning for a *file* is a simple string match that matches anywhere in the line. As such, the **REM-** command should **never** appear before the equal sign. Placing the **REM-** command incorrectly effectively neutralizes its effect.

Arguments:

None.

Returns:

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

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

10.2.4 RUN0

Table 10.12: Scheduling Control

Command sample	Description
RUN0 *schedule*	run selected schedule

Action:

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.

10.2.5 STAR

Table 10.13: Start Scheduling

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

Action:

Suspends scheduling until the indicated *start time*.
In other words, active *mod* schedules will only start running after the specified time.

Take note that this does not affect startup files (i.e. **INI=**, **TEST=**, **MAS=**, and **ANN=**). These all run outside the scheduling loop leaving them unaffected.

Arguments:

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

Time spec must be complete (hours, minutes, and seconds)!

The stored time is one second before that given on the command line.
This enables scheduling just before the specified time so we don't miss the first scheduling event.

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.

10.2.6 IDLE

Table 10.14: 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.

10.2.7 STAT

Table 10.15: 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.

10.2.8 CONF

Table 10.16: Hardware Configuration

Command sample	Description
CONF	Configure Hardware Subsystems

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.

Use the **CONF** command with no arguments to dump the keywords available to this command.

Table 10.17: 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
XTAL	Document RF synthesizer crystal
CTL	DRA818/SA818 Power Down Enable
PTT	Carrier Enable (external transceiver)
TX_ENA	Transmit enable
DB_PWR	Daughterboard Power Switch
AM	AM Mode
CW	CW Mode
FM	FM Mode
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
6PF	SI5351 reference Xtal load 6pF
8PF	SI5351 reference Xtal load 8pF
10PF	SI5351 reference Xtal load 10pF
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.

This sub-command selects default T0..T5 timing parameters. This will overwrite any previous selection, so place changes to the T0..T5 timing parameters **after** this sub-command.

ICS307

SubCommands

RF Synthesizer select.

Not implemented (only on the 102-73181-0 board).

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. The reference crystal load capacitance may also be adjusted should the need arise.

This sub-command selects default T0..T5 timing parameters. This will overwrite any previous selection, so place changes to the T0..T5 timing parameters **after** this sub-command.

DRA818/SA818

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.

This sub-command selects default T0..T5 timing parameters. This will overwrite any previous selection, so place changes to the T0..T5 timing parameters **after** this sub-command.

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 is non-functional. Serial control of the handie-talkie requires 102-73181-10 or later circuit board.

This sub-command selects default T0..T5 timing parameters. This will overwrite any previous selection, so place changes to the T0..T5 timing parameters **after** this sub-command.

Control of the handie-talkie, other than simple **PTT** and **audio** will require updates to the operating software. This would tend to imply that a specific handie-talkie would be used with all stations.

XTAL

SubCommands

Document the RF Synthesizer crystal frequency.

This defualts to 20.0MHz, the same crystal frequency as used by the zNEO.

This only documents the alternate frequency, the crystal frequency isn't used by the zNEO software.

nMA

SubCommands

SI5351 clock output drive.

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

The **nMA** sub-command and **CLKn** sub-command may be combined:

esav INI=8MA,CLK0

The **CONF SI5351** comand provides a default setting for this value. Changes must be provided following the **CONF SI5351** comand.

nPF

SubCommands

SI5351 reference clock crystal load capacitance.

The SI5351 has a programmable load on the reference crystal that can be adjusted with these configuration controls. A default of 6pF is selected by the **CONF SI5351** command.

The **nPF**, **nMA** and **CLKn** sub-command may be combined:

esav INI=8MA,CLK0,8PF

The **CONF SI5351** comand provides a default setting for this value. Changes must be provided following the **CONF SI5351** comand.

Changing from the default of **6PF** to **8PF** will lower the carrier frequency by 4KHz to 5KHz.

Changing from the default of **6PF** to **10PF** will lower the carrier frequency by 8KHz to 10KHz.

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.

The **CONF SI5351** comand provides a default setting for this value. Changes must be provided following the **CONF SI5351** comand.

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 10.1: 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.

AM and CW

SubCommand

AM is an alias for CW.

Use the form: **-AM** to return to normal FM operation.

This flag requires the use of the 102-73181-36 power amp board.

This flag tells the CW generator (i.e. the CW interrupt service routine) to toggle the **TX_ENA** net along with the **TONE_ENABLE** net. This causes the transmitter to operate in an *AM* mode, where carrier esacapes only during the *dit* and the *dah*. This is essentially operating as an HF code transmitter.

You can make use the **TONE 0** command to disable FM modulation.

This type of operation is intended primarily for a code-only style of operation. The use of **TALK** commands works, with the carrier being enabled prior to each audio fragment. The audio will be choppy as the carrier is disabled following each audio fragment.

FM

SubCommand Added in the 3.94 software release

FM is an alias for -AM.
Use this to return to normal FM operation.

No help text appears when issuing the **CONF** command with no arguments.

T0=nn

Timing SubCommand; RF subsystem power enable

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

See figure 4.15 on page 42.

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

The DB_PWR signal is asserted to turn on U81 and U91 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.

This also has a minor effect on the 102-73181-28 module. This module has and additional switch to control power to the RF amplifier although the default setting of 10 is more than adequate.

T1=nn

Timing SubCommand; RF subsystem powerup

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

See figure 4.15 on page 42.

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 within a transmitter group will, in effect, shift all messages within the group of fox transmitters by the same time. This will be most evident when a mix of DRA818/SA818 RF modules, and SI5351 RF clock synthesizer are used. Although the SI5351 does not require the long delay that is needed by the DRA818/SA818 RF modules, the long delay will not interfere with SI5351 operation.

This timing delay may also be increased if the first dit of the signon message (from the **BEGN** command) is not heard.

A new value of 150 seems to help:

esav INI=CONF,T1=150

T2=nn

Timing SubCommand; RF enable

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

See figure 4.15 on page 42.

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

The **TX_ENA** signal is asserted in this state (this appears as **PTT*** on the DRA818/SA818 daughter board).

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

The default value for this delay was increased in the V3.64 software release to give the RF subsystem time to stabilize before the first chip (the letter **E**) of the signon message is sent.

T3

Message Delivery

See figure 4.15 on page 42.

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.

See figure 4.15 on page 42.

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.

See figure 4.15 on page 42.

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.

10.2.9 TOYC

Table 10.18: TOY Clock Charge

Command sample	Description
TOYC	DS1672 setup report
TOYC *<keyword>*	reconfigures the DS1672

Action:

With an argument, sets up the DS1672 charge control in the same manner as the older TOYE command did.

With no argument, reports on the DS1672 current settings.

Control register patterns 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**"

10.2.10 TIME

Table 10.19: 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.

None (to set system time).

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.
The TOY clock is polled until the seconds field changes before updating the system time to achieve a bit better timing resolution.

We assume the clock setting utility synchronizes with the host system to achieve good sub-second synchronization.

If a time string is present in the command, it is converted to binary and copied to the system time field and the DS1672 time registers are updated.

The RTI (ie.. sub-seconds) are set to zero.

The TOY clock seems to take a bit of time to come to life (or other issues that affect startup). To get a reliable time read at startup, issue the **TIME** command twice with an intervening delay. More discussion and an example may be found in section 4.8.2 on page 52).

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.

10.2.11 TIRP

Table 10.20: Time reporting command

Command sample	Description
TIRP	synchronize, beep, and verbalize seconds
TIRP <*n*>	repeat count
TIRP **A N S**	verbalization enables

Action:

Verbalize a time synchronizatiuon message.

Synchronize to zero RTI at *seconds(0..59) mod 5*.

Send a beep at the currently selected audio tone *(the beep occurs at the start of the second)*.

Verbalize the current seconds *(that run 0 to 59)*.

Arguments:

Numeric Repeat Count.

A enables sending the **V_TIRP** audio clip when the **TIRP** command starts running.

N enables verbalizing the 2 digits of the seconds field.

S enables verbalizing the **V_SEC** audio clip.

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 intended to be placed into the **ANN=** message to provide a synchronization check of the TOY clock as the fox transmitters are turned on and placed at the begining of a hunt.

It is **not** intended as a general command to be used inside of a normal message. The delay before the beep may be up to 5 seconds as the fox transmitter waits for a scheduling point to occur.

You may choose to turn transmitters on, one-at-a-time prior to walking out in the field to check that the TOY clock is working as expected and we aren't low on battery power.

The **V__TIRP** audio clip announces that a time synch operatio is about to be performed. A *heads up* if you will.

The **V__SEC** audio clip we expect to say "seconds".

10.2.12 D525

Table 10.21: ICS525 management command

Command sample	Description
D525	Dumps the ICS525 Divisors

Action:
ICS525 setup diagnostic.

Arguments:

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.

Note the status report that is delivered when the ICS525 handler is not loaded:

```
STS13,01* Handler_D525 (cmd_ics525_dummy.c*)   0.03 Sec
```

10.2.13 EPOC

Table 10.22: Time management command 3

Command sample	Description
EPOC *argument*	

Action:

Set the timezone offset (to display in terms of local time).

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;
```

–or–

```
system_time  = Unix_Time % 864000;
system_time += 86400;
```

The shifting by one day keeps the *EPOC*h offset from causing the resulting time to go negative.

By way of example, the command for operating with *Central Daylight Time* is **EPOC -5.0**.

10.2.14 CALL

Table 10.23: Setup Callsign

Command sample	Description
CALL *callsign*	FCC assigned callsign

Action:

The supplied text is saved in the callsign variable.

This callsign is sent at the end of every message.

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>

10.2.15 NAME & NICK

Table 10.24: Setup Nickname

Command sample	Description
NAME *name*	tactical callsign
NICK *name*	tactical callsign

Action:

The supplied text is saved in the name/nickname variable.

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>

10.2.16 TONE

Table 10.25: CW Tone Control

Command sample	Description
TONE *Hz*	Audio tone frequency select
TONE 0.0	Disable audio tone

Action:

Sets the audio tone frequency.

The frequency selection is constrained to be between 250Hz and 2.5KHz.

The tone generator can be disabled by calling out a frequency of 0.0.

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.

10.2.17 CWPM

Table 10.26: 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 10.27: 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.

10.2.18 FREQ

Table 10.28: Transmit carrier frequency control

Command sample	Description
FREQ *freq*	Operating Frequency

Action:

Save the Transmitter Operating Frequency.

This new frequency selection will then be loaded into the RF subsystem when the **BEGN** command is executed.

The frequency selection is **not immediate** to avoid causing operational problems!

Were the frequency selection to take effect immediuately, such as when actively transmitting, you would lose contact with the fox transmittER.

Arguments:

Frequency, in MHz.

The action performed depends on the transmit element selected in the **CONF** command.

The **FREQ** command can make use of supplemental frequency tables stored in the FRAM. See section 9.15 on page 158 for details of encoding the supplemental frequency table.

DRA818/SA818

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 (typically from a table that was loaded into FRAM).

The result of the lookup are visible in the **STAT** command reports and a failed lookup is reported in the *stsxx,xx** report.

ICS525

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 ICS525 setup patterns are extracted and saved for use when the **BEGN** command configures the ICS525.

The result of the lookup are visible in the **STAT** command reports and a failed lookup is reported in the *stsxx,xx** report.

EXTERN

External Handie Talkie

The frequency string will need to be massaged into the appropriate setup commands for the handie-talkie in use. The V3.70 software does not handle any handie-talkie.

Frequency Limiting

The **FREQ** handler does very minimal limits checking to try to keep frerquency selection sane. All operating frequencies have been migrated to external tables, so effect any required limits by loading tables with valid entries.

10.2.19 FOFF

Table 10.29: Transmit carrier frequency offset

Command sample	Description
FOFF *freq*	Frequency Offset

Action:

Save the Transmitter Frequency Offset.

Arguments:

Offset, in KHz.

This command simply documents the offset frequency we are operating with at the moment. It does not affect operation of the frequency synthesizer!

The *fox_simple* program generates a command for the **INI** sequence. It appears as **INI=FOFF**,<*offset*> and indicates the offset used to generate the external frequency table. After we measure the actual operating frequency of the synthesizer we generate an external frequency table to operate at the target frequency and note the applied offset with this command.

10.2.20 5351

Table 10.30: SI5351 Control

Command sample	Description
5351	register key and parameters

Action:

SI5351 diagnostic command.

This command is used when developing setup tables for the SI5351. We also use this command to inspect the internal 5351 tables.

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 10.31: 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

Dump the SI5351 registers through the serial channel.

SDUMP

Full Register Dump

Dumps the entire 256 byte address space of the SI5351 through the serial channel.

TABLE

Frequency Table Dump

Dumps the *internal frequency table* for the SI5351 through the serial channel.

Around V3.76 the internal table had the crystal frequency offset removed. This implies that the *internal frequency table* will not operate the SI5351 at the correct frequency. For proper operation you must characterize the offset error and load an external table to correct for the error.

Around V3.80 the *internal frequency table* is loaded with a small number of points spread across the 2M band to allow you to measure the error at a point that is useful for your fox hunt.

RESET

SI5351 PLL Reset

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.

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 HI" message until a keyboard character arrives.

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.

Loaded into the SI5351 when the LOAD sub-command occurs or when the **BEGN** occurs.

XTALT

Diagnostic

For diagnostics, the crystal oscillator output is connected to the CLK0 output pin.

I2CT

I2C transaction test

Generates continuous I2C transactions addressed to the SI5351.

The SI5351 must respond to I2C transactions with an acknowledge!

I2CR

I2C transaction test

Generates continuous I2C read transactions addressed to the SI5351.

I2CW

I2C transaction test

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.

10.2.21 BEGN

Table 10.32: Begin Message Traffic

Command sample	Description
BEGN	
BEGN *SILENT*	Suppress ID
BEGN *SILENT tone*	Suppress ID and send audio

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.

This modifier should result in an unmodulated carrier. The bandwidth here representing the reference clock noise (i.e. short term crystal stability).

SILENT tone modifier.

This modifier suppresses the code ID that is sent as part of the BEGN sequence and enables the tone generator at the specified frequency (in KHz). This in a testing modifier intended to aid in analyzing the performance of the output filter on the motherboard.

This modifier should result in the carrier being spread, the bandwidth representing the carrier modulation.

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 send **CQ CQ CQ** and the stored callsign (see the **CALL** command in section 10.2.14 on page 185).

The current system clock is saved for use by the *DONE* command.

10.2.22 CODE

Table 10.33: Generate Morse Code

Command sample	Description
CODE *text*	

Action:
Sends a CW message.

Arguments:

Message text.

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.

As of version 4.08 the **CODE** <CALL> and **CODE** <NAME> perform as expected.

10.2.23 TALK

Table 10.34: 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 the RIFF/WAV header in the audio file. If this is a simple waveform file with no header, TALK directory entry will have the sampling rate and sample count. The the sample 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 FLASH memory. As mentioned previously, a raw wavefoprm clip may have the sample rate and sample count explicitly sotred in the directory entry. Sample rates of 4K/sec, 5K/sec, 8K/sec, 10K/sec and 16K/sec are implemented.

10.2.24 WAIT

Table 10.35: Simple Wait

Command sample	**Description**
WAIT *n.n*	Wait specified time (in decimal seconds)
WAIT *p/o*	Synchronous Wait (period/offset in seconds)
WAIT SWITCH x	Wait for switch input state
WAIT PHOTO	Wait for Photocell input change

Action:

Waits for specified time.
Waits for synchronization time.
Waits for switch state.
Waits for photocell change.

Arguments:

Time in seconds (decimal).

Scheduling point in period/offset notation.

Wait for external event trigger.

Contact closure (for SWITCH).

Change in analog channel (for PHOTO).

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.

V3.80 add a synchronous wait feature.

This is an immediate synchronization with the schedule specified in the argument. The slash (/) delimiter was just something that is convenient. No whitespace between the period and offset values.

When combined with the **CONF CW** configuration command, you can run transmitters within a group in close synchronization with eachother (the **CONF CW** will cause carrier to drop during the waiting period).

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.

10.2.25 CHRP

Table 10.36: Chirp Emulator

Command sample	**Description**
CHRP *tone per dur rep*	V3.73 and prior (do **not** use!)
CHRP *tone per off dur rep*	V3.75 and later
CHRP *file per off delay rep*	added in V3.84 and later

Action:

Chirping tracker emulation.

Arguments:

Audio file (same as in the TALK command).
Audio tone frequency (float KHz).
Chirp period (int seconds).
Chirp offset (int seconds).
Chirp duration (float seconds).
Repeat count (int).

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 (i.e. seconds). That is to say the chirp will occur every *period* seconds. The chirp tone duration is set by the *Chirp duration* parameter (duration is fractional).

An update starting at V3.75 changes the scheduling method to a synchronous model. In this model, the period and offset granularity are one second, exactly like of main scheduler.

Specify *per* and *off* just as with any other schedule.
The command halts the zNEO between chirps to reduce power (similar to the main loop does).

Argument 1: frequency/filename

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.

Version 3.84 adds the capability to send out an audio file in place of a simple tone. Any audio fragment can be sent here, but it is intended to allow an actual chirp to be stored in FLASH.

Argument 2: Period in seconds

Period (fractional seconds V3.73 and earlier).
Period (integer seconds V3.75 and later).

Specifies the repeat period of the "*chirp*". The *chirp* repeates every *Argument 2* seconds.

Argument 3: Offset in seconds

Offset (integer seconds)

Specifies the offset (in seconds) into the repeat period.

Argument 3/4: Duration in seconds

Chirp duration (fractional seconds).

Specifies the time (in seconds) the chirp is sent. Carrier is active during this period.

Version 3.84 changes the meaning of this field when sending out an audio file. This field specifies the settling or warm-up time after the RF section is enabled before the audio starts streaming out.

Argument 4/5: count

Repeat count (integer).

The chirp is repeated this many times.

A negative value will cause a timetage to emit before each chirp.

With the synchronous scheduling method, it becomes possible to setup a schedule where a group is mixed at a much finer granularity (requires V3.76 or later).

Consider five transmitters operating with a 10 second chirp period. Just like we schedule message traffic staggered through a 5 or 10 minute cycle, we can setup offsets of 0, 2, 4, 6 and 8 seconds to have the chirp group operate without overlapping.

You have to deal with the signon traffic that is required every 10 minutes, but the signon can be abbreviated (using only the BEGN command) with all stations carefully aligned at the begining of a 10 minute cycle.

The argument 5 counts will all be different to force the end of message traffic to align (i.e. use DONE SILENT to suppress ending ID message when the BEGN message occurs immediately).

The V3.84 version adds the ability to an audio file in place of a simple tone burst. The power switching works the same, so when configured for interrupted carrier operation we still remove carrier when not sending audio. Also note the duration field is redefined to avoid the need to embed a quiet section at the begining of the audio file to allow the RF section to stabilize.

10.2.26 DONE

Table 10.37: Done with message traffic

Command sample	Description
DONE	
DONE *SILENT*	Suppress ID

Action:

End of message traffic.

Arguments:

SILENT modifier.

This modifier suppresses the code ID that is sent as part of the DONE sequence. This in a non-FOX related modifier, is should not be used when transmitting over the air as you will not send the required station ID at the end of your message.

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 in section 10.2.14 on page 185) to satisfy FCC rules and a CW *SK SK SK*.

10.2.27 BATC

Table 10.38: Battery Report CODE

Command sample	Description
BATC *keys*	<modifier>,<key>,<V-limit>

Action:
Battery Report in code.

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 10.39: BATC Modifiers

Modifier	Description
B	A/D sample collected before BEGN
A	A/D sample collected after BEGN
E	Encode value using T/E

Table 10.40: BATC Keywords

Channel Key	Description
V	Battery Voltage and trip point
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 **V** modifier is used to report voltage and it may include a voltage trip point. If the measured voltage falls below the trip point, the code message that reports the voltage will include **SOS SOS** to indicate we are in a critical battery state.

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 switch-mode 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. Power plot may be found in section 4.10.2 on page 61.

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.

10.2.28 BATV

Table 10.41: Battery Report VOICE

Command sample	Description
BATV *keys*	modifiers

Action:
Battery Report Voiced.

Arguments:

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 ignored prior to V3.87. Starting at V3.87 we get an SOS in code just like the BATC command is the battery voltage is below the specified limit.

10.2.29 BATR

Table 10.42: Battery Report Text

Command sample	Description
BATR *flag*	Battery Report

Action:
Battery Report (textual).

Arguments:

A flag character **I** for coefficients table dump.

Returns:

sts47,00* Handler_BATR (cmd_battery.c*) 7.5V 7.31445e-03 *coefficients table line*

STSnn,00* response with status and command execution time along with the battery state report.

RDY00,0n* response with current stack pointer value and current system time.

This command generates a battery state report showing current voltage and current as seen in the Fox Transmitter.
This is intended to be used to characterize battery capacity.

You may find it useful to load the current UNIX time into the Fox Transmitter prior to a battery performance run. This will result in the current time, rather than a truncated time, to appear in the battery report.

10.2.30 MODS

Table 10.43: 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: Sn

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 10.44: 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

10.2.31 MODC

Table 10.45: 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: Sn=

Schedule number.

The schedule is expressed in 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.

10.2.32 TALK Filesystem directory commands

Table 10.46: TALK Filesystem directory

Command sample	Description
esav TALK=*name start length rate*	Name, start location, length, and rate
esav TALK=*name start*	Name and start location

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 not 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.

10.2.33 ESAV text

Table 10.47: 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.

Overflow:

Writing too many records to the FRAM will not cause overflow. The FRAM handler will not write past the end of the physical FRAM present in the system, rather overflow data is discarded.

The smallest FRAM device the software will deal with is 64Kb which is room for 256 records. This size device is marginally small for managing a large or complicated hunt. Whatever commands overflow the FRAM are, of course, lost.

10.2.34 EDMP text

Table 10.48: FRAM control EDMP

Command sample	Description
EDMP	
EDMP *key*	Entries that have *key*

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*!

10.2.35 EDID

Table 10.49: 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.

10.2.36 ERAS

Table 10.50: FRAM control ERAS

Command sample	Description
ERAS *n*	remove record (i.e. REM-)
ERAS *start stop*	remove multiple records
ERAS *DEV*	erase device

Action:

Remove the FRAM record by inserting a **MT**** command in place of the existing command.

Remove multiple records by specifying a range of records.

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 (or range of numbers) rewrites the specified record(s) with a **MT**** command.

This leaves the remainder of the FRAM file system visible.

DEV

Entire Device

Erase the entire FRAM device.

Using ***fox_binary*** to avoid using the **HERA** command.

There is no need to erase the FRAM when using the **HERA** command as the *fox_binary* loader completely bypasses the (somewhat primitive) file system. The *fox_binary* loader will place a zero record following the last command that is loaded to properly mark the end of active records.

This behaviour will be particularly useful if you have audio clips loaded into the FRAM on older boards that do not have a seperate FLASH for storing waveform data. The *fox_binary* utility leaves that waveform data undisturbed.

10.2.37 EZER

Table 10.51: FRAM control EZER

Command sample	Description
EZER *n*	erase record (ZERO)
EZER *start stop*	erase multiple records (ZERO)

Action:

Erase a single record from FRAM.

Erase multiple records from FRAM.

Arguments:

Specifying a single record number zeroes the specified record.

Specify a start record and stop record to zero a range of records.

This leaves the remainder of the FRAM file system *invisible*.

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.

10.2.38 ETAB

Table 10.52: 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 10.53: 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.

The FLASH_AAI types will not work with the binary loader!

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.

10.2.39 HEND

Table 10.54: FLASH control HEND

Command sample	Description
HEND	

Action:
Find start of empty flash device.

Arguments:
None.

Returns:

sts01,nn* response with address of empty space in the flash device.

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

The scan occurs with a 4K stride from end of device towards begining.

This provides a means of finding unused areas of a large flash device.

10.2.40 HERA

Table 10.55: FLASH control HERA

Command sample	Description
HERA BLOCK 0x<*start*>	erase record
HERA ALL	device erase

Action:
Erase region of FLASH device.

Arguments:
BLOCK Erase block: Block Starting Address.

ALL Erase device: No Arguments.

Returns:

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

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

This command has **no sanity checking** to prevent an unintended erase operation.

This command is **in**sensitive to the **CONF VOICE** setting (see section 10.2.8 on page 172). 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** or **block erase** command to the device.

This makes the flash device look *dead* until the **chip erase** or **block erase** operation has finished. Sending any flash commands will return a **BUSY** message until the device reports it is ready.

For the block erase variant of the command; the address argument is passed, unchanged, along with the block erase command (0xD8) to the FRAM device. The user must consult the datasheet to correctly form the address to erase the desired area of the device (typically 64K)

Also take note that the address argument is in hexadecimal!

10.2.41 HDMP

Table 10.56: 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 10.2.8 on page 172). 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.

10.2.42 H56K/H115

Table 10.57: H115/H56K

Command sample	Description
H56K	Set bit rate to 57,600 b/S
H115	Set bit rate to 115,200 b/S
H56K PROG	Set bit rate to 57,600 b/S, FRAM loader binary mode
H56K WAVE	Set bit rate to 57,600 b/S, FLASH loader binary mode
H115 PROG	Set bit rate to 115,200 b/S FRAM loader binary mode
H115 WAVE	Set bit rate to 115,200 b/S FLASH loader binary mode

Action:

Switch bit rate to 57,600 bits/second (control port defaults to 57,600 b/S).

Switch bit rate to 115,200 bits/second (control port defaults to 57,600 b/S).

Switch bit rate to 57,600 bits/second and operate in binary transfer mode.

Switch bit rate to 115,200 bits/second and operate in binary transfer mode.

Arguments:

None.

Returns:

RDY00,00* response with current stack pointer value and current system time. Operating at the new bit rate.

The clock divisor at this bit rate was off by up to 5%. The data communication between host and target would not have been reliable on versions prior to 3.93 (changed divisor by 1 count).

Older method to speed up the audio loading process (prior to V4).

First, switch the Fox Transmitter over to the higher speed.

V3.68 and prior use H56K to switch to 57,600 b/s.

```
H56K
sLask.jcn
```

V3.69 and later use H11 to switch to 115,200 b/s.

```
H115
sLask.jcn
```

For either case, the STS and RDY response are, of course, garbled (hence the *sLask.jcn* or some other unInTelligible garbage) due to the mis-matched bit rates.

Next switch the monitoring terminal over to match the new bit rate.

```
.../halo_term -b57600 -SFOX2X
.../halo_term -b115200 -SFOX2X
.../halo_term -SFOX115
```

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 around 50 milliseconds (i.e. the **-c50** on the call line).

```
.../fox_simple -b57600 -SFOX2X -c50 -t10 -ffox_73181_rxxa.hex
.../fox_simple -b115200 -SFOX2X -c50 -t10 -ffox_73181_rxxa.hex
.../fox_simple -SFOX115 -c50 -t10 -ffox_73181_rxxa.hex
```

The higher bit rate reduces the transmission time, of course, 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) This may allow some overlap to occur, although the time required to program a 32 byte line in the flash may ultimately limit speed.

H115 WAVE and H56K PROG

The last forms of the command, with either the **WAVE** or the **PROG** modifiers, switches over to the binary loader. The **WAVE** modifier is used to program FLASH memory. The **PROG** modifier is used to program FRAM memory.

When loading FLASH using the binary protocol the target area of the FLASH must be erased before using the **WAVE** modifier.

When loading FRAM using the binary protocol it is not necessary to perform an erase operation (**HERA dev**) to clear FRAM.

The binary loader bypasses the file system, directly writing records to FRAM. One record following the last record sent is cleared to ZERO. A dump of FRAM will show only the content that was just loaded.

The binary protocol is identical for loading either memory device.

The behavior of this form of the command is to switch bit rates to the specified rate and then switch the serial port *Interrupt Service Routine* over to a binary mode in order to perform a fast download to FLASH or FRAM memory.

Currently, downloading InTel HEX files is rather slow, taking 30 to 60 minutes to load the audio file system with waveform data. This hook switches to a binary type of processing where the *ISR* expects fixed length binary packet data to be sent by the host.

Loading time with the binary handler is reduced to a few minutes for a 500KB FLASH load and a few seconds for a 300+ command FRAM load.

You must keep the download within the address limits of the target device. Neither the *fox_binary* utility nor the software in the fox transmitter do any bounds checking when loading using the binary protocol.

Software versions later than **Version 4** are required to deal with this feature.

10.2.43 :hex

Table 10.58: InTel HEX Record Load

Command sample	Description
:hex *record*	InTel HEX record up to 32 bytes long

Action:
Load FLASH device.

Arguments:
InTel HEX record (up to 32 bytes of data)

Data **must** be naturally aligned (FLASH device restriction).

> The flash device data alignment requirements are not enforced by the flash loader, data records **must** be generated that do not violate alignment requirements.

Returns:

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

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

Valid Record Types:

TYPE 0: INTEL_RECORD_TYPE_DATA

Memory image data.

TYPE 1: INTEL_RECORD_TYPE_EOF

Indicates this is the last record in a group.

TYPE 3: INTEL_RECORD_TYPE_EXTENDED_SEGMENT

Address bits 19..4

This address data is added to the address in the data record.

TYPE 4: INTEL_RECORD_TYPE_EXTENDED_LINEAR

Address bits 31..16

This address data provides the upper 16 bits of the data record address.

Checksum:

The *InTel HEX* loader validates the checksum at the end of the hex record before it is loaded into FLASH. An invalid checksum will cause the record to be rejected.

Overflow:

Writing too many records to the FLASH **will** cause overflow. The FLASH handler does not check for addresses that extend past the end of the memory device.

Expect problems to occur if the *InTel HEX* Load is too large for the device. At a minimum, expect the first audio file to be corrupted.

The FLASH decice requires a seperate erase operation to restore the memory array to all **1**s. All we can write to the memory array is **0**s. This behavior indicates the an overflo write will be catastrophic and require a device erase to recover.

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 52494646501000005741564566 6D74201000000001000100A00F0000A00F0000 4F
:20 0020 00 0100080064617461 2B10000808081808 07F8080808080808080808080808080 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. These *whitespace* characters are ignored, they ar **not** required.

This command is sensitive to the **CONF VOICE** setting (see section 10.2.8 on page 172).. See comments in section 10.2.41 on page 212.

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.

10.2.44 HALT

Table 10.59: 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.

This 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.

10.2.45 STOP

Table 10.60: 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).

This 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.

10.2.46 TEST

Table 10.61: Test Suite

Command sample	Description
TEST	Misc. test commands

Action:
Pass control to test handler

(Many looping tests will stop with a keypress)

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 10.62: 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

10.3 Sample Sequences

Example code for setting up a FOX Transmitter.

10.3.1 Initialization

This code runs following reset, hence the *INI=* tag.

When both the **TEST** and the **MAS** jumpers are present, we skip this setup step.

Table 10.63: Sample Sequence 1

esav INI=TIME
esav INI=WAIT,0.5
esav INI=TIME
esav INI=EPOC,-5.0
esav INI=CALL KC0JFQ
esav INI=NAME FOX20
esav INI=CONF DRA818
esav INI=MODS S1 300 0
esav INI=MODS S2 300 150
esav INI=TONE 1.6
esav INI=CWPM 25
esav INI=FREQ 144.150
esav INI=STAT

10.3.2 TIME

Time Set. This loads the 32 bit time from the TOY clock into the system time field. Take note that the command is issued twice to get a reliable load from the DS1672. This is required in order for multiple units to operate synchronously.

10.3.3 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.

10.3.4 CALL/NAME

Sets the callsign and nickname for the CQ and SK messages the precede and follow message traffic.

10.3.5 CONF

Tells software the RF subsystem in use. traffic.

10.3.6 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.

10.3.7 TONE

Sets the audio tone frequency.

10.3.8 CWPM

Sets the code chipping rate to 25 WPM.

10.3.9 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.

10.3.10 STAT

Displays the system status on the USB serial port.

10.4 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.

When both the **TEST** and the **MAS** jumpers are present, we skip this setup step.

Table 10.64: 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

10.4.1 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.

10.4.2 CWPM

Set the message to a comfortably high speed to get the message sent quickly.

10.4.3 BEGN

Send the *CQ* message with our callsign.

10.4.4 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 **dah** characters representing volts and the number of **dit** characters representing tenths.

10.4.5 DONE

Send our callsign and the *SK* message.

10.4.6 FREQ

Change over to the operating frequency for the group.

10.4.7 TONE

Change over to the target audio frequency.

10.4.8 CWPM

Change over to the target chipping rate.

10.4.9 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.

10.5 Sample Sequences

Example code for setting up a FOX Transmitter.

10.5.1 Schedule 1 Sequence

This code is the *S1* sequence.

Table 10.65: 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

10.5.2 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.

10.5.3 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.

10.5.4 WAIT

Leaves an unmodulated carrier on and waits the specified number of seconds.

10.5.5 CODE

This sends a few characters of the message. Each storage record holds up to 25 bytes of message text.

10.5.6 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.

Chapter 11

Practical Sequencing

An exercise in maintaining your sanity.

11.1 Sequences

Sequences 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 5.3.1 on page 99.

11.1.1 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=TIME                    "tickle DS1672"
INI=WAIT 0.5                "allow DS1672 to wake up"
INI=TIME                    "get time from DS1672"
INI=EPOC,-5.0               "CDT, offset to local time from ZULU"
INI=NAME,xxx                "tactical call"
INI=CALL,xxx                "station callsign"
INI=CONF,SI5351             "Hardware we're running on"
INI=CONF,CLK2,8MA           "SI5351 output configuration"
INI=FREQ,144.150            "Common announcement frequency"
INI=FOFF,-14.0              "Document SI5351 frequency offset"
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"
. . .
INI=FOFF,-14.0              "note the frequency offset"
```

The **INI=FOFF,-14.0** notes the frequency offset used to generate the external frequency table. The V3.72 release incorporates a small table, without offset, to allow the fox transmitter to be characterized before selecting the appropriate external frequency table.

The **FOFF** command notes the offset in use having no other functional effect on the tranmsitter system. The specified offset shows up in the output of the **STAT** command.

11.1.2 TEST=

Test Jumper *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 should 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"
```

We are free to put anything useful in this *file*. It is perfectly permissible to **EZER** and **ERAS** records, rewrite and add new records, test and repeat to achieve the your desired testing goals.

11.1.3 MAS=

Master Jumper *sequence*

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
```

The labeling of this jumper is a left-over from the days when we tried to implement a time synchronization methodology that could be used in the field. With the introduction of the 102-73181-10 hardware, this the synchronization feature has been deprecated.

The 102-73181-10 hardware moves the configuration channel (i.e. the USB serial port) to the externally accessible port and repurposes the 2nd. serial port (i.e. the time synchronization network) for accessing the DRA818/SA818 RF module.

The **MAS=** file functions identically to the **TEST=** file. You are free to make any convenient use of it.

11.1.4 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"
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=RUN0,S0                 "Enable the ZERO schedule"
ANN=REM-                    "Remark"
```

This *feature* was implemented to make life easy for the hunt organizer.

In the example we announce the stations callsign as a sanity check. It should be what you expect it to be, or something is *off*.

We also verbalize the station nickname, again so that the hunt organizer can verify which unit is being placed.

Battery condition is also vocalized. We have the capability so we use it to, again, provide the hunt organizer with useful information concerning the condition of the battery in the transmitter.

So when you drop a unit, you can wait until the last moment to turn the unit on and still know that it is operating as the desired station and that the battery has sufficient power to last through the entire hunt.

11.1.5 ID=

Identification Records.

These records are **not** necessary for the fox transmitter to function properly.

These are strictly documentation records for the benefit of the operator. Some of these are generated by the utilities that load FRAM and FLASH memory to document which files were created the load and the time when the load was created of the file loaded.

Here is an example of the **ID=** records found in the FRAM of **FOX36** at the time this section was written:

```
  0: ID=LT,FOX-Binary-Loader V1.4
  1: ID=LT,2025-04-21T19:46:25
  2: ID=FR,FOX36_KC0JFQ.log
  3: ID=FR,2025-04-21T19:46:20
  8: ID=FL,talk_73181_2025_TREK.fox
  9: ID=FL,2025-04-21T18:44:04
```

*The records in the **TALK=** directory are here*

```
104: ID=FL,SIZE,0xC1B80
```

Files ***INI=***, ***MAS=***, ***TEXT=***, ***ANN=***, *are here*

```
185: ID=S2,talk_73181_2025_TREK.hex
194: ID=S3,talk_73181_2025_TREK.hex
208: ID=S4,talk_73181_2025_TREK.hex
```

The external frequency table records are here

```
372: ID=FR,SIZE,0x2E20,373
```

This listing has the record numbers for each line. It should also be obvious that records in this filesystem all start with **ID=**. As illustrated here, these records are scattered throughout the **ID=** filesystem.

The *fox_binary* utility.

The *fox_binary* utility dramatically reduces load time both command sequences and audio files. We use it for loading both FRAM and FLASH.

This utility switches the system to operate a high speed binary protocol that reduces traffic volume sent over the communications link. When the load is complete, the utility switches the system back to normal interactive operation.

You can easily avoid conflicts by running the *fox_binary* utility and the *halo_term* utility from one terminal window. This prevents both from running at the same time. This prevents *halo_term* from capturing the **ACK** response required by the *fox_binary* utility.

The *fox_binary* utility with command sequences:

The *fox_binary* utility places the first four **ID=** records that indicate that the *fox_binary* utility effected the load.

The first record (number 0) simply notes that the FRAM was loaded by the *fox_binary* utility. The *fox_binary* version string also shows up to allow us to track changes to what records we should expect to see inserted into FRAM by the *fox_binary* utility.

The first date (record number 1, **ID=LT**, where **LT** stands for Load Text) indicates the *fox_binary* utility version and when the *fox_binary* utility was run to load the FRAM. The date tag will change every time the FRAM is updated using the *fox_binary* utility.

Record number 2 is the file (log file from *fox_simple*) that was used to load the FRAM. This file has all the translated commands ready to copy directly into FRAM. The *fox_simple* utility is used in a shell script to build the *FOX36_KC0JFQ.log*.

The second date (in record number 3) is the file creation date from the *FOX36_KC0JFQ.log* file. It the modification date stamped by the file system on the host system.

The *fox_binary* utility also tracks how much of the FRAM is in use and places a final **ID=FR,SIZE** record showing the current use.

The next free address and the record count are shown in record number 372.

The *fox_binary* utility with audio files:

When loading an audio file, the *fox_binary* utility inserts records into the **ID=** filesystem that describe the audo file.

These are the **ID=FL** records, the **FL** simply indicating FLash memory information.

The *fox_binary* utility notes the source audio hex file in record number 8. The audio hex file modification date is in record number 9. And finally the size is saved in record number 104 of the audio file in the FLASH memory.

We see the next free address in the FLASH (always 128 byte aligned)

Records inserted from within sequences.

You are free to use the **ID=** filesystem to store any informaiton you find useful. You are limited only by the size of the FRAM and how the search speed affects the timing of operations.

In our example above, we also have records that are inserted at the begining of the **S2=** (record number 185), **S3=** (record number 194), and **S4=** (record number 208) sequences. Were you to dump (using the **EDMP** command) the contents of the FRAM on a working system you would see any sequence documentation records adjacent to the sequence they are in. The utilities take no part in creating them.

The examples here are to illustrate the these three command sequences require the *talk_73181_2025_TREK.hex* file be present in FLASH for audio to be produced.

Note the correlation in the filename from the **ID=FR** record and the **ID=S2** record. The TALK directory matches up with the audio file so we should be able to make use of the extra voice clips in this file.

11.1.6 Sequence Fault Recovery

Master Jumper and Test Jumper

In the event that the sequence commands loaded into FRAM prevent the Fox Transmitter from operating, you can bypass all initialization steps by installing both the **MAS** jumper and the **TEST** jumper.

With both jumpers installed you have access to the command decoder before any commands in the FRAM are executed. At this point you can erase FRAM and reload using the *fox_simple* loader. You can also directly load a new image (without the need to erase the current contents) using the *fox_binary* loader.

With both jumpers installed the contents of the FRAM are left undisturbed. You can inspect the FRAM using the **EDMP** command. The **EDMP** command can be used with a match string if that aids in your search for the problem.

...

11.1.7 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)"
```

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.

11.1.8 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)"
```

11.2 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.

11.3 Time Synchronization days(s) prior to foxhunt

The following may all be checked days before the fox hunt.

To synchronize time before the fox hunt, you connect to your desktop (or laptop) and run the *fox_simple* program to set the TOY clock.

The 102-73181-5 and all 102-73161-* boards are accessed using the USB port, so the case must be opened. The 102-73181-10 boards may be accessed using the externally accessible 3.5mm jack. A 50 Ohm load is required when powering-on the unit to avoid damage to the RF subsystem.

One by one connect each Fox Transmitter and run the *fox_simple* program to set the TOY clock. Typically the unit is configured for the hunt (i.e. the **TEST** and **MAS** jumpers are empty), so you will need to wait for the announce message before sending the time update.

The TOY clocks have probably not been trimmed, so they will drift apart, but the error is small enought that setting the time a day or two before the hunt should not cause a problem.

11.4 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 for novice trackers, 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* as 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.

11.5 Carrier Frequency

The system is also designed to allow selecting the carrier frequency.

Nominally, you would select a common carrier frequency in the **INI=** file so all units send their announce message on one common frequency.

You then set the operating frequency at the end of the **ANN=** file. This allows multiple hunt groups to share one **INI=** file while keeping the all of the announce message traffic on a common frequency.

Bear in mind that the **FREQ=** command may appear in the schedule. The software architecture allows you to change the frequency right in the middle of a message!

11.6 Accessing the USB port

The USB port is mechanically inaccessible because it is inside the box on older boards (102-73181-5 and earlier). This, then, requires 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. Simply operate the unit on the bench with the cover removed.

The 102-73181-10 units move the the serial port over to the connector that was the network time port to provide access without having to open the box.

The network time port was never used, it having been reallocated to the DA818/DRA818 tranceiver modules.

This move eliminates the need to open the box to update the TOY clock, the FRAM, or the FLASH.

11.7 Accessing the 3.5mm port

Experience with the ICARC club hunts indicate that the 3.5mm port doesn't seem to collect debris. You may consider covering the port during the hunt with a bit of tape or a label.

Amazon lists a quantity of *3.5mm Earphone Plug Cover*s for a few dollars. These should provide a convenient means of keping debris out of the serial port jack.

Chapter 12

fox_simple Utilities

This utility is used to load the FRAM and FLASH in the fox transmitter.

This utility is written in plain-old C to run on a Linux box.

One of the *fox_simple* utility functions is to send time updates to the target system. Time updates are synchronized on the host system to occur as seconds change, that is to say when subseconds on the host system are very close to zero.
The *fox_simple* utility will update time synchronously.

12.1 fox_simple utility: Command Line Arguments

Descriptions of the command line arguments used with the fox_simple utility.

This is not an exhaustive list, only showing the switches used to load sequence files and audio files.

12.1.1 fox_simple -S <*port*>

Serial Port Name.

This argument provides the *nickname* of the serial port we will use to talk to the target fox transmitter. The *nickname* is used to lookup the serial port operating parameters (bit rate, parity, etc.) and the full path in the /dev filesystem to the serial device.

In our examples that follow, the **FOX2X** refers to an FTDIchip USB to serial cable. The setup table specifies, for this device, the bit rate of **57,600**, 8 data bits, and no parity.

12.1.2 fox_simple -c <*delay*>

Change in inter-line delay.

The default delays imposed after each line delivered to the target may be shortened using the switch. Shortening the inter-line delay will, of course, speed up the load operation.

12.1.3 fox_simple -t *<time generation>*

Reduce the precision of the time delivered to the target.

Without this switch, the full UNIX time field is sent to the fox transmitter. Using this switch reduces the time to cover the specified number of days so that manually looking at the time value is more human friendly.

12.1.4 fox_simple -C *<Callsign>*

Substitution value for **'call'** in the downloaded file.

Set the station callsign in the target file (see the -f switch) by substituting occurrences of the **'call'** string with the *<Callsign>* supplied here.

This is to allow a single setup file to be used for a large group of fox transmitters.

12.1.5 fox_simple -N *<Nickname>*

Substitution value for **'name'** in the downloaded file.

Set the station nickname in the target file (see the -f switch) by substituting occurrences of the **'name'** spring with the *<Nickname>* supplied here.

This is to allow a single setup file to be used for a large group of fox transmitters. The station name, in particular, should be unique for each transmitter.

12.1.6 fox_simple -Q *<freq>*

Substitution value for **'freq'** in the downloaded file.

This is used to set the target operating frequency for the fox transmitter. When setting up a multi-group hunt, we can still make use of a single setup file using this substitution to separate groups by frequency.

This substitution usually occurs at the very end of the **INI=** setup.

12.1.7 fox_simple -R *<schedule>*

Substitution value for **'run'** in the downloaded file.

Again, we manage the setup of a large group of transmitters by providing a schedule substitution switch. The operating schedule for transmitters in a group will have a common period, but unique offsets. This is used to provide a unique offset for each transmitter in a group.

12.1.8 fox_simple -A *<offset>*

Substitution value for **'ftab'** in the downloaded file.

This is an accommodation to manage external frequency tables for the SI5351 where your set of transmitters all have slightly different reference crystal operating frequencies.

We generate a frequency setup table for each possible offset and use this switch to select the appropriate frequency table to load.

12.1.9 fox_simple -X *<key=value>*

Generic substitution where **'key'** is replaced by **value** in the downloaded file.

This increases the flexibility of managing multiple transmitters in a group using a single confiscation file.

12.1.10 fox_simple -f *<filename>*

Input file name.

This is the master sequence file.

12.2 fox_simple utility: Loading Sequence Files

Listing 12.1: FOX2X_KC0JFQ-8

```
#  /home/wtr/Radio/halo_term/fox_simple                                    8
#                              -fFOX2X_KC0JFQ.fox -lFOX2X_KC0JFQ.log       9
#                              -Xchrp1=6,0 -Xchrpfrq=1.0                   10
#                              -Xsched=S0 -Xtone=1.0                       11
#                              -Xstooge=120 -Xchirp_up=0                   12
#                              -Xchirp_dn=0 -Xruns6=600,300                13
#                              -Xsynth_dev=SI5351 -Xsynth_set1=8MA         14
#                              -Xsynth_set2=CLK0 -Xtalk_file=talk_73181_rxxk.  15
    ↪ fox
#                              -Xsx_stooge=SX_STOOGE.fox -Xspare1=not      16
#                              -Xspare2=used -Xbatvc=BATV                  17
#                              -CW0JV -NFOX24                              18
#                              -R360,180 -Q144.285 -Afreq_5351-08.fox      19
```

Here is a typical call line to generate a log file for fast loading into one of the transmitters used by the *Iowa City Amateur Radio Club*. These example lines are extracted from the **FOX2X_KC0JFQ.fox** file with the line numbers reflecting their place in the file. They all use the same FTDIchip serial cable to communicate with the fox transmitters which are plugged into the external 3.5mm jack.

Most of the command line switches match, with just three unit specific switches.

The units all operate using the club callsign (**W0JV**), so the **-C** switches are the same. The inter-line delay and time truncation switches are also all the same; no need for anything unique here.

The units all must have unique **nickname**s. The **nickname** will be used by the fox transmitter to select an audio clip to verbally identify the fox transmitter as part of its sign-on message.

The target operating frequency, **144.325**, is the same on all units. We provide it here as the **FOX2X__KC0JFQ.fox** file is used to load multiple groups that operate of unique frequencies.

The scheduling parameters, supplied through the **-R360,*** switches must be unique and properly spaced. In our example here, take note of a 6 minute period, with each unit offset by and additional 60 seconds.

If the transmitted message fits within the allotted 60 seconds we should never hear two transmitters operating at the same time.

The next variable switch argument is the frequency offset of the reference crystal oscillator. These offsets were measured during unit commissioning and are listed here to allow the appropriate external frequency table to be selected.

The last switch on the call line is the name of the sequence file we will load into the fox transmitter. With the above listed substitutions, each transmitter ends up a few sequence commands that are unique to the transmitter. Each transmitter uniquely identifies itself and runs at the correct time. We have also loaded the correct external frequency table to let the SI5351 generate the correct carrier frequency.

12.3 fox_simple utility: Loading Audio Files

The audio file system directory is loaded as part of the main sequence load. The *'talk_file'* string is replaced by the file specified on the command line that builds the download image.

Listing 12.2: FOX2X_KC0JFQ-30

```
#  Limited voice storage                                    30
#                                                           31
# //#include talk_73181_1.fox                              32
#include 'talk_file'                                        33
```

Example command line to load HEX file:

```
fox_simple -SFOX2X -c50 -t10 -f/home/wtr/WAV/fox_73181_r4k.hex
fox_binary -SFOX2X  -a/home/wtr/WAV/talk_73181_2025_TREK.hex
```

Loading the audio file system is somewhat simpler in that we typically would not bother to tailor the contents of the audio file system for each unique fox. Rather we simply provide a complete set of audio clips to each transmitter with many of the voice fragments going unused. Any given transmitter would use only one nickname and ignore all the rest.

Note that the load image for the *fox simple* utility must **not** have any blank lines!

If the *fox simple* utility encounters a zero-length line, it stops processing the file at that point.

You may, if you want empty space in the file, simply pad with **REM-** lines. The *fox simple* utility recognizes this keyword and skips sending it to the target system,.

We can speed up the load process by boosting the serial rate to **115,200** bits per second although this isn't strictly necessary. The call-line on line 22 may be used directly without altering the serial rate of the fox transmitter. If the call-line on line 23 is to be used, the *H115* must be issued to switch the fox transmitter serial rate.
Switching to the higher rate reduces the time spent sending serial bit to the fox target, but will not change the time the fox transmitter requires to decode and store the hex record. The **-c 50** switch seems to allow time for the hex record decode and the time required to program a 32 byte *chunk* of the flash device.

There is some overlap that is possible as the fox transmitter will buffer incoming traffic while waiting for the flash device program operation to complete. This is open-loop, so the **-c** switch ultimately controls the programming time allowed for the flash device.

We will always update the TOY clock when **fox_simple** runs, so we specify the truncation switch (**-t 10**).

The input file in this case is the InTel HEX file for the audio file system. It should have been generated with 32 data octets per line; anything longer will overflow the input buffer. Shorter lines (i.e. 16 octet lines) will program the flash device correctly while doubling the required loading time.

12.4 fox_binary utility: Fast Binary Loader

The time required to load the FLASH memory using the text command interface will get excruciatingly slow as the size of the audio image increases. This gets frustrating if you are working out a new set of audio clips or loading a large skulk of transmitters.

To address this, the version V4.00 release of the Fox Transmitter operating software adds a binary loader feature. This capability is invoked using either the **H115 WAVE** or the **H56K WAVE** command.

Version V4.01 expands on this by expanding this to provide for loading the FRAM in the same manner. For loading FRAM, you will need the log output from the *fox_simple* utility (run without the **-S** argument).

Also bear in mind that updating the FRAM using the *fox_binary* utility does not require the FRAM to be erased prior to reloading. This speeds up loading of the FRAM be eliminating steps.

The *fox_binary* utility operates a simple binary protocol over the command link that is used to load operating sequences. The **H115 WAVE** or **H56K PROG** command (note the **WAVE** or **PROG** modifier/argument) will switch to the desired bit rate and run the binary protocol.

The following command line flags operate just like those from the *fox_binary* utility.

12.4.1 fox_binary -h

Display a short help text.

This lists the command line arguments processed by the fox_binary utility.

12.4.2 fox_binary -d

Increase Debug Level.

This allows debug text to be emitted by the fox_binary utility.

12.4.3 fox_binary -F

Switch bit rate to 225,200 bits/sec.

This sends the **H115** command to switch over to the binary protocol at the elevated bit rate.

After the last record is sent, the fox transmitter and the **fox_binary** utility both switch back to operating at 57,600 bits/sec

12.4.4 fox_binary -S *<port>*

Serial Port Name.

This argument provides the *nickname* of the serial port we will use to talk to the target fox transmitter. The *nickname* is used to lookup the serial port operating parameters (bit rate, parity, etc.) and the full path in the /dev filesystem to the serial device.

In our examples that follow, the **FOX2X** refers to an FTDIchip USB to serial cable. The setup table specifies, for this device, the bit rate of **57,600**, 8 data bits, and no parity.

12.4.5 fox_binary -a *<file>*

Load Audio File (FLASH).

The audio file loader expects InTeL HEX records. Additional text may be intersperced in the file and it will be ignored.

The *-a* flag will take either **filename.hex** or **filename**. The **.hex** will be added if it's not there.

12.4.6 fox_binary -f <*file*>

Load Sequence File (FRAM).

The sequence command loader expects a log file processed by the *fox_simple* utility. All substitutions will have been performed by *fox_simple* to produce this file. It is a list of the actual commands that are to be sent to the target.

WE still need the fox_simple utility to perform all the substitutions. This results in a log file, fragments of which are shown in listing on page 241.

Sample *-f* file fragment from somewhere in the **FOX22** image:

Listing 12.3: FOX22_KC0JFQ.log

```
072 esav TALK=TS4_BOSTON 386688                    71
073 esav TALK=2K1_H_9000 395264                    72
074 esav TALK=2K1_GD_EVE 415616                    73
075 esav TALK=2K1_CHESS2 427264                    74
076 esav TALK=2K1_ENJOYA 439552                    75
077 esav TALK=2K1_JUST_MOM 447616                  76
078 esav TALK=2K1_MSG_4_U 471168                   77
```

The first column is, in effect, the record number in the FRAM file system (starting at 0). This field is used to calculate to address in FRAM where this record will be loaded.

The second column is the command to save the record to FRAM which whould have been used by the *fox_sdimple* utility. It will not be used by *fox_binary*.

The final text group is the command that will be saved to FRAM. In our case, this is exactly what need to be stored in the FRAM.

Looking at the **FOX2X_KC0JFQ.fox** which was used to generate this log file, note that the unit-unique fields have all been replaced with their target values. Only the substitutions that are performed by the Fox Transmitter remain (i.e. <CALL>and <NAME>).

12.4.7 fox_binary utility: Inserted Records

The *fox_binary* utility inserts several records into the load during the download process. These inserted records provide some details of the download operation and the file that was loaded into the target.

The inserted documentation records are all in the **ID=** file (so may be inspected using a **EDMP ID=** command).

This record looks more-or-less like this:

```
ID=LT,FOX-Binary-Loader V1.4
```

The **LT** key simply indicates this is a **L**oad **T**ext record.

The *FOX Binary Loader* version is included as the first inserted record. This record serves to document that the FRAM file system was loaded using the binary loader.

The local time when the *FOX Binary Loader* was run to load the FRAM file is shown on the second line.

This record looks more-or-less like this:

```
ID=LT,2025-04-20T14:10:24
```

The final group of records show where and when the file contents came from, that is the file that sourced the records in the FRAM filesystem.

The next inserted record is the filename of this load image file. It looks similar to this:

```
ID=FR,FOX36_KC0JFQ.log
```

The **FR** key indicates this is information about the FRAM memory device.

We also publish the file creation time, as this is invariably useful at some point.

This record looks more-or-less like this:

```
ID=FR,2025-04-20T14:00:48
```

The date format follows ISO 8601 where most significant numbers are on the left.

The time should be expressed in **l**ocal **t**ime.

Finally, as the last record inserted by the *FOX Binary Loader* is the memory allocation of the FRAM.

This record looks like this:

```
ID=FR,SIZE,0x2800,324
```

The allocated size of the FRAM in bytes (in hexadecimal) and the number of records (in decimal).

Externally inserted records

External processing scripts may insert records into the **ID=** filesystem for the same reason that the *FOX Binary Loader* does. It is expected that external utilities will follow the conventions described here. In particular, keep the date format consistent!

We can also insert records into the **ID=** filesystem in any of the 10 sequences to indicate which audio file is required for proper operation.

Externally inserted records, *fox_simple* utility

The audio processing scripts also inject records into the file where the TALK= directory is stored. These are also intended to appear in the **ID=** file system.

We identify the InTel HEX file used to load the FLASH memory with the first record

This record looks close to this:

```
ID=FL,talk_73181_2025_1.fox
```

This reports the file name that the audio build scripts use to save the master memory image.

The date when the script runs is next in line. Although this timestamp is collected in the shell script, the resulting HEX file is recreated at the same time keeping the timestamp consistent.

This record looks like this:

```
ID=FL,2025-04-20T14:00:34
```

Note the appearance of the timestamp matches that produced by the *FOX Binary Loader*.

An, as the final record inserted by the audio shell script, we note the next free page in the FLASH device.

This record appears as:

```
ID=FL,SIZE,0x75700
```

The data value used here is provided by the *audio file utility*. It is the result that is passed along to the next invocation of the *audio file utility* as the start address for the next audio clip.

Externally inserted records, sequence files

As operating scripts are developed that make use of unique voice clips, there may arise a dependency on a particular audio file.

This audio file records look like to this:

```
ID=S2,talk_73181_2025_TREK.hex
ID=S3,talk_73181_2025_TREK.hex
ID=S4,talk_73181_2025_TREK.hex
```

This documents the need for the *talk_73181_2025_TREK.hex* waveform file for correct operation of these sequences. These records are manually included as part of the sequence that requires a specific voice clip for proper operation.

12.4.8 fox_binary utility: Protocol Details

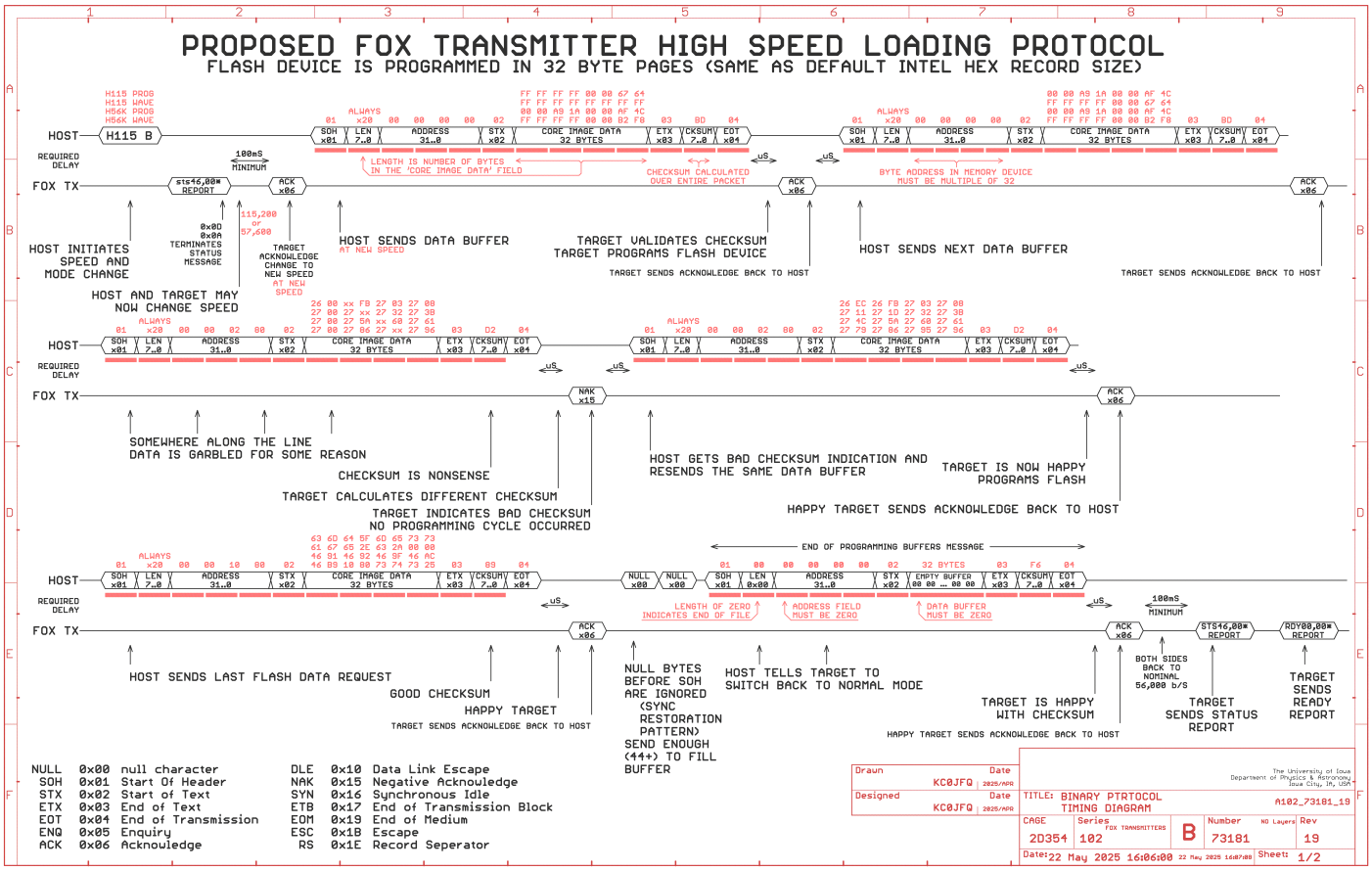


Figure 12.1: Binary Protocol

This is now a working proposal for the binary packet protocol to dramatically improve the speed with which the audio file system can be loaded.

Either the **H115 WAVE** command or the **H56K WAVE** command is used to run the binary loader. The InTel HEX format loader is retained and continues to operate as before.

After the **H115 WAVE**/**H56K WAVE** command is sent, The target (i.e. the Fox Transmitter) sends back a normal (lower case) status report and then waits. The target switches to the targeted bit rate.

In practice, the target waits around 100mS between the *sys46,00** report and sending the initial acknowledge. This delay allows time for the host system to collect that status report and then switch over to the new bit rate.

The target, now operating at the new rate, responds with the single acknowledge ***ACK*** *(0x06)* character to indicate it is ready to proceed with the binary load.

Host formats a buffer with 32 bytes of image data and the ten bytes of overhead for a total buffer size of 42 bytes. This fixed length buffer is sent to the target. The target must validate the checksum and the the length field, before proceeding. Assuming validity checks pass, the image data is then sent along to the FLASH device for a page program operation.

In binary mode, the behavior of the **ISR** changes to deal with the packetized data, but beyond the **ISR** level, the call to retrieve a buffer remains pretty much the same. Reading a packet in binary mode always uses a fixed length of 42 bytes to simplify the code in the interrupt routine.

Once the packet is received (by the ISR) and forwarded, packet validation can begin.

If packet validation fails the target will return a ***NAK*** *(0x15)* which informs the host of the damaged packet.

Error recovery from a ***NAK*** response is up to the host. The target doesn't keep track of anything from packet to packet. The host can re-transmit or continue as it sees fit.

If validation succeeds the target will return an ***ACK*** *(0x06)* which tells host to transmit the next buffer.

Packet transfer and packet processing are **not** overlapped. The flash write routine waits for completion (from the target memory device) before responding.

To program the FLASH device, the byte address in the packet needs to be down-shifted to form a packet address for the programming routine (it seems to want a 32 byte record address rather than a byte address).

This ends up doing a bit of useless shifting, but the zNEO reduces this shift operation to just a few instructions.

The flash write routine polls the FLASH device for write complete, so the write time is closed loop. Once write is complete, target returns an ***ACK*** *(0x06)* character to indicate host may proceed with the next buffer.

This handshake proceeds as the entire audio file system is loaded.

The address field, being 32 bits, covers a 32Gb (4GB) flash device. This covers all devices that are currently available in the 8 pin package.

At the end of the load, the host then signals the target to return to normal command processing. The packet sent to accomplish this is formatted exactly like any other packet but with a length of zero and an address of 0x00000000.

The out-of-bound length value is used to trigger the return to normal operations.

The target will respond to this buffer just like any other by sending the ***ACK*** character if the checksum is valid or a ***NAK*** character if bad. Host will have to take appropriate recovery action if receiving a ***NAK*** (such as a single retry).

When the target returns a ***NAK*** as result of a bad packet checksum, it will not return the ISR to normal operation. A valid packet (with a valid checksum) is required to return to normal command processing. A simple reset or power-cycle can be used to recover from this type of hang.

After sending the final ***ACK***, the target will poll the UART, waiting for the transmit shift register to clear and then switch bit rate back to the standard rate of 57,600 b/S.

Host, upon reception of the ***ACK*** character, should switch its bit rate back as well.

The target, now back to the nominal bit rate again, waits for around 100 mSec before sending any additional traffic to allow host time to transition back to the nominal bit rate.

The target handler is now finished processing the load and can proceed with that additional traffic. The target will format a normal **sts** response and pass it back to the main loop where the **sts** is *upcase*'d and we then see **STS** and then the normal **RDY** message.

We can then carry on with normal interactions.

Checksum calculation is simple addition across the **entire buffer**, from the ***SOH*** through to and including the ***EOT*** This is to simplify the calculation at the target. The target will accumulate a sum across the whole buffer and the checksum should be zero.

The only checking that occurs at interrupt level is the search for the initial ***SOH*** character. When the ***SOH*** arrives, the ISR switches to loading the remaining 41 bytes of the packet.

Once the 42 bytes of the packet have arrived they are passed up to the binary loader for validation and further processing.

The binary loader directs the ISR into and out of the binary load mode. So the **H115 WAVE/H56K WAVE** command triggers the transition to binary loader mode and the **null** record (at the end of the binary load) triggers the transition back to normal command mode.

Fixed Length Packets in binary mode?

By fixing the packet length at 32 data bytes (and 10 overhead bytes) the **interrupt service routine** instruction path length is reduced.

The goal is to allow the ISR to operate up at 115,200 bits/seconds without dropping characters.

The fast path through the ISR is for buffering the body of the packet, the 41 bytes following the ***SOH*** character. The ***SOH*** character requires special processing so the ISR checks for additional data in the UART receive buffer register before leaving the ISR. This aims to eliminate the possibility of encountering an overrun condition due to this ***SOH*** processing.

The UART seems to like 2 stop bits when operating in the binary mode. The host utility should setup the host channel to accommodate.

The UART bit rate divisor at 115,200 is not exact. Using 2 stop bits allows receive receiver a bit more time resynchronize.

The Binary Loader processing audio (waveform) **files**

Not much special happens here. Loading the audio waveforms expects 32 byte InTel HEX records in the source file. Non HEX records are ignored. The Binary Loader adds a NULL record at the end to transition the fox transmitter back to normal command mode.

The Binary Loader ignores non HEX-Records and will process data through to the end of the file. Blanks lines or thost that don't start with a colon are ignored.

Records are sent to the target Fox Transmitter as they are decoded. We assume the host system is fast and will introduce minimal delays when decoding the InTel HEX-Records.

...

...

The Binary Loader processing command files

There is a bit of special handling when loading command files using the binary loader (using the **-f** flag with a filename). The utility inserts 4 records into an ***ID=*** file at the beginning of the load.

They are placed into the ***ID=*** file to make displaying the a bit more convenient using the **EDMP ID=** command.

An example of the four lines:

ID=FOX-Binary-Loader V1.3	Identifies that the command text was place by the binary loader
ID=LT,2025-04-18T22:04:04	This is the date when the binary loader was run
ID=FR,FOX27_KC0JFQ.log	This is the name of the command log file
ID=FR,2025-04-17T20:04:34	This is the date when the command log file was created

The second line updates every time the Binary Loader is run to load the fox transmitter.

The third line is the modification date of the command log file.

These text lines must fit within the fixed 32 byte record stored in the FRAM. Only the text on line 3 changes in size. All the other lines have fixed text fields. Contents change but the length does not.

The third line is taken from the command line that is used to run the Binary Loader. If the filename exceeds 29 characters it will be truncated to fit within the 32 byte record in FRAM.

...

The Binary Protocol

Binary Protocol Command: Initiation

Activate the binary protocol with either the **H56K** command or the **H115** command. In either case you must supply either the **PROG** modifier or the **WAVE** modifier.

If the modifier is missing, the bit rate change will be applied but the binary protocol will not be activated.

Note that programming FLASH memory requires a manually initiated erase operation (**HERA all**) to pre-clear the flash. FLASH is a write-0 device, so it must be cleared to all 1s before you can successfully write to it.

Programming FRAM, on the other hand, does not require the erase operation. The FRAM is, after all, just a RAM. The RAM may be written to either a 1 state of a 0 state. The binary loader, when loading FRAM, writes a zero record after the last command record written to the FRAM. This zero'd out record serves as an end-of-file delimiter.

If you write a short image to the FRAM (i.e. one that is smaller than what is currently in FRAM) you will need to clear the FRAM (**ERAS dev**) if you plan to add any records following the load for the FRAM file system to behave as expected. Since we only clear **one record** following the load, there could be records from a previous load that will magically appear after writing to the zero-filled record.

Binary Protocol Reply: Initiation

Target responds with a status message to indicate that the changeover to binary protocol has started:

```
sts46,00* binary_loader.c* ready
```

Target now delays for roughly 100 milliseconds to give host time to switch to the new bit rate.

Following the delay, target sends an acknowledgment character ***ACK*** *(0x06)* at the selected bit rate.

There is no requirement that we change bit rates at this point. This serves to provide a means of operating at a higher bit rate to speed up the loading process.

Binary Protocol Command: data packet

Assuming the host correctly received the ***ACK***, data packet flow can begin.

Host formats a **fixed length** packet with 32 bytes of image data. The length field is (almost) always 32 (0x20), the exception being the last packet sent. The address field is the **byte address** of the first byte of data in the packet.

The checksum is calculated across the entire packet and should give a zero result. This is an 8 bit sum with an end-off carry.

Once properly formatted, the host sends the packet to the target.

Binary Protocol Reply: data packet

Upon packet reception, target must verify the checksum. A bad checksum (i.e. non-zero) should be indicated by sending a negative acknowledge ***NAK*** *(0x15)* telling the host a problem occurred. Target will then prepare for the next packet.

A valid checksum (where is adds up to zero) allows the target to proceed on to the programming operation. Note that we don't send an ***ACK*** back just yet as we don't know if the programming step will be successful.

A good checksum allows target to proceed with the programming operation. The **PROG** or **WAVE** modifier has already set the flash routines up to address the correct external memory device, so here we simply call the flash write routine and then poll the external memory device status register until the programming operation indicates success or a timeout occurs. A timeout will result in the target sending back a ***NAK*** *(0x15)* reply.

In the nominal case, we get an operation complete indication from the external memory device and then send the ***ACK*** back to the host.

This *data packet* transaction repeats for each packet needed to program the target memory device.

Binary Protocol Command: end packet

After all of the programming packets have been transferred from host to target, the host will tell the target to return to normal command operation by disabling the binary protocol.

The indicator to the target that the return to regular operation is to occur is a packet with a length field set to zero. The address field and data field should also be set to zero to keep things pretty. Currently the target doesn't depend on this but the *fox_binary* utility does zero out unused fields in the packet.

The checksum is calculated as expected and placed in the packet. The host may then send the end packet on to the target and wait for a reply.

Binary Protocol Reply: end packet

The target validates the checksum, as always, and responds if an error occurs. If the checksum is valid, target will notice the length is zero a react accordingly.

Target will acknowledge that the packet was correctly received with an ***ACK***.

Target will then delay for a bit, again on the order of 100mSec or so, to give host time to switch back to base bit rate as the target switches itself back to the original rate (typically 57,600 b/S).

After the delay normal command termination activities occur. The binary loader will format a status reply string and pass it back to the main control loop.

Command Protocol

In the main control loop the status string will have the "sts" string (at the beginning of the string) uncased to "STS" and sent out over the communications channel. This provides the indication that the fox transmitter is now processing commands and is operating back at the original bit rate.

The main control loop will then send out the "RDY00,00*" message to indicate readiness to process the next command (exactly as with any other command).

...

Chapter 13

fox_clock Utility

This utility is used to set the TOY clock in the fox transmitter.

This utility is written in plain-old C to run on a Linux box.

In the madness of getting setup the night before a hunt, getting all the clock set can be a bit of a nuisance. In normal trim, the fox transmitter will send out its sign-on message (on 144.150MHz) and then sit idle until its assigned transmit window occurs.

We can use the host computer to watch the status traffic coming out of the serial port and inject time commands when the transmitter becomes idle. The status traffic coming from the fox transmitter is explicitly engineered to allowed computer control of such things.

The program logic is pretty well fixed, all we do is set the TOY clock. We simply wait for the **RDY00** prompt from the fox transmitter and then send out a few setup commands.

The first command is a **TOYC NONE** to make sure we're not providing charge current through the DS1672. The external charge control circuit is the only current supplied to the backup battery.

We the synchronize with the system time on the host system. This synchronization waits for the sub-seconds to reach zero before the *fox_clock* utility sends the time setup message to the fox transmitter.

Following synchronization, we obtain time from the host system, truncates it (the fox transmitter scheduler ignores days), places the time into a **TIME** command and sends to the target.

The host synchronization methodology assumes that the host system is lightly loaded. Under light load, the *fox_clock* utility runs without interruptions to get the target clock consistently set.

13.1 fox clock utility operation

There are only 3 arguments that the *fox_clock* utility will recognize.

13.1.1 -h

Help text, terse though it may be.

13.1.2 -d

Increase debug level.

Not of much use.

13.1.3 -S USB port

Serial port nickname.

The *fox_clock* utility uses the *halo_term* library so the nicknames available are the same as the *halo_term* nicknames. par

13.1.4 -l transmitter log filename

This file holds the performance data for all the fox transmitters that are updated. When loading the clock, the utility will also list all the units that have been loaded today.

This argument, when given by itself will dump the logfile using todays date (i.e. does not expect traffic from the fox transmitter).

13.1.5 -m label csv filename

This is the file used by the *fox_label* utility. When the *fox_clock* utility runs it collects the current condition of the battery, which is of interest to the *fox_label* utility. The battery readings for the *fox_label* utility are updated.

The *fox_clock* utility also captures the operating frequency and updates it in the label file.

label csv filename backups

The *fox_clock* utility does not overwrite the existing file, rather it renames it and creates a new file. A date string is generated and appended to the filename applied in the rename operation.

An example directory listing fragment following a run of the *fox_clock* utility:

```
-rw-r--r--. 1 wtr wtr    4723 Feb 21 15:58 fox_label.csv-2025_Feb_21T16:08:40
-rw-r--r--. 1 wtr wtr    4723 Feb 21 16:08 fox_label.csv
```

The old file has the **-2025_Feb_21T16:08:40** stuck onto the end of the filename and then we simply create a new file (**fox_label.csv**).

13.2 Normal Use

Invoke as follows:

./fox_clock -SFOX2X -lfoxlog.csv -mfox_label.csv

The utility then waits for traffic from the fox transmitter.

So, connect up the target, run **./fox_clock -SFOX2X** and then turn the target fox transmitter on.

I find it useful to have a 50Ω load on the transmitter and an H.T. set to 144.150MHz. You hear the sign-on message and then the *fox_clock* utility sets the clock and prints a status report.

A logging facility leaves a trace of the information gathered from the fox transmitter when the time is updated. The battery voltage, in particular, can be reviewed after updating all the transmitters to make sure there is sufficient battery for the hunt.

You can also review the sequence that will be run during the hunt. Check that it what you expect (typically **S0=**).

Sample Log File:

Listing 13.1: foxlog_1.txt

```
2025-May-22T21:13:21,"V4.04","W0JV","FOX27",144.325,"S0",8.67-V,30-mA,8.35-V,120-mA,-5Hrs,Off:4   1
2025-May-22T21:14:07,"V4.04","W0JV","FOX25",144.225,"S6",8.92-V,38-mA,8.62-V,122-mA,-5Hrs,Off:4   2
2025-May-22T21:14:49,"V4.04","W0JV","FOX29",144.325,"S0",7.83-V,42-mA,7.35-V,146-mA,-5Hrs,Off:5   3
2025-May-25T15:33:37,"V4.04","W0JV","FOX21",144.225,"S6",8.02-V,24-mA,7.60-V,121-mA,-5Hrs,Off:6   4
2025-May-25T16:28:09,"V4.04","W0JV","FOX25",144.225,"S6",8.90-V,37-mA,8.61-V,122-mA,-5Hrs,Off:4   5
```

This is data stored in the *foxlog.csv* file. The collected data is then used by the label utility (see section 16.9 on page 290).

13.3 Extracted Reports

Sample Report:

Listing 13.2: foxlog_2.txt

```
 S/W Version: "V4.04"                                                                                1
    CALLSIGN: "W0JV"                                                                                 2
    NICKNAME: "FOX30"                                                                                3
   Frequency: "144.325"                                                                              4
    RUN Slot: "S0"                                                                                   5
  System Off: "Now:68338D3A" "Off:6"                                                                 6
 System Time: "Sys:68338D58" "TOY:68338D58" Epoc:"68400"  -5 Hrs                                     7
  Idle Power: 7.89-V 47-mA                                                                           8
Active Power: 7.28-V 150-mA                                                                          9
fox_clock.c             main/1652 Decode Last Detail Record                                          10
fox_clock.c             main/1660 CMD: sort -t"," -k4,4 -k1,1 < foxlog.csv > sorted_foxlog.csv       11
fox_clock.c             main/1664 Report Todays Loads                                                12
fox_clock.c    Todays_Loads/1098 "FOX21": 2025-May-25T15:33:37,"V4.04","W0JV","FOX21",144.225,"S6    13
    ↪ ",8.02-V,24-mA,7.60-V,121-mA,-5Hrs,Off:6
fox_clock.c    Todays_Loads/1098 "FOX25": 2025-May-25T16:28:09,"V4.04","W0JV","FOX25",144.225,"S6    14
    ↪ ",8.90-V,37-mA,8.61-V,122-mA,-5Hrs,Off:4
fox_clock.c    Todays_Loads/1098 "FOX30": 2025-May-25T16:36:25,"V4.04","W0JV","FOX30",144.325,"S0    15
    ↪ ",7.89-V,47-mA,7.28-V,150-mA,-5Hrs,Off:6
fox_clock.c             main/1670 CMD: rm sorted_foxlog.csv                                          16
fox_clock.c             main/1681 UPDATE Labels: fox_label.csv                                       17
fox_clock.c      Last_Write/1296 CMD: mv fox_label.csv fox_label.csv-2025_May_25T16:36:25             18
```

This is a sample report produced by the **fox_clock** utility.

This is an update on **FOX30** as indicated by the *NICKNAME:"FOX30"* detail line. We can verify that the callsign (in this case **W0JV**) is set and we will operate at 144.325MHz (as intended).

The TOY clock offset is captured as the Fox Transmitter is started. The clock utility records local time and the time from the second **TIME** command in the **ANN=** sequence. These two times are differe4nced and reported on the *System Off: "Now: 68338D3A" "Off:6 "* line. Here we see that the Fox Transmitter was ahead by 6 seconds *"Off:6"*.

Battery levels are adequate to operate the hunt, the active power is above 7.2V (*Active Power: 7.71-V 149-mA*).

After acquiring the data from the fox transmitter, the log file is sorted by nickname and date to be listed in order. Only the details record from the current day are printed. In this example 4 of the set of 12 transmitters have been updated so far.
Current draw seems a bit high, showing 54mA when idle and drawing 150mA when transmitting. Finally, notice that we are operating CDT as indicated by the **-5Hrs** at the end of the line. This is the *Epoch* field from the fox transmitter. This *Epoch* field must be set correctly for the **STAR** command to operate correctly.

13.3.1 Handler_TIME

Data from the TOY clock used to verify things loaded correctly.

We use the second occurrence of this command to determine the TOY clock drift.

13.3.2 Handler_EPOC

This reports the current time-zone offset setting.

13.3.3 Handler_BATR

This is the voltage and current at the time the command was issued. If your sign-on sequence is properly configured, it will have a battery report before the transmitters is turned on as well as when the unit is transmitting.

13.3.4 Handler_CALL

The fox transmitter callsign.

13.3.5 Handler_NAME

The fox transmitter nickname.

13.3.6 Handler_FREQ

The fox transmitter operating frequency.

This will appear several times in the **ANN=** sequence, so the last one seen is reported.

This last frequency should be the operating frequency (not the announce frequency).

13.3.7 Handler_RUN

The fox transmitter active sequence.

The last (or very nearly the last) command in the **ANN=** sequence should activate the schedule the transmitter will operate on.

Nominally, our operating sequence would be the **S0=** sequence. If that is expected, and something else shows up, you have the opportunity to correct this before being embarrassed at the hunt.

13.4 Clock setting shell script

Shell script used by the author to automate clock setting operations.
This script allows for quickly updating all units and keeping track of the data they produce.

```
#!/usr/bin/bash
#
#
#        -D "Todays Date"
#
FOX_CLOCK=/home/wtr/Radio/halo_term/fox_clock
SERIAL=FOX2X
#
echo ready
echo "*********************************************************************************"
echo "******************************** FIRST UNIT *********************************"
echo "*********************************************************************************"
while :
  do
    $FOX_CLOCK -S$SERIAL -l foxlog.csv -mfox_label.csv
    echo "*********************************************************************************"
    echo "******************************** SWAP UNITS *********************************"
    echo "*********************************************************************************"
  done
```

Using a simple looping construct in the script allows you to plug in a fox transmitter, turn it on, and wait for the announce message to be sent. After the announce traffic is sent, the *fox_clock* sets the clock and updates the *foxlog.csv* and the *fox_label.csv* files.

We then wait for another to be plugged in and switched on.

The *fox label utility* should be run **after** the *fox clock utility*. This has the latest battery condition available when the **fox_label_A_chk.*** files are produced.

13.5 Reviewing Battery Condition

After setting time in all units, you can review the collected by running the *fox_clock* utility without the USB port argument.

Invoke as follows:

./fox_clock -lfoxlog.csv -mfox_label.csv

Get something like this:

```
fox_clock.c           main/1373 Restore Last Detail Record from "foxlog.csv"  37 records
fox_clock.c           main/1384 Decode Last Detail Record
fox_clock.c           main/1392 CMD: sort -t"," -k4,4 -k1,1 < foxlog.csv > sorted_foxlog.csv
fox_clock.c           main/1396 Report Todays Loads
fox_clock.c    Todays_Loads/ 863 "FOX24": 2025-Feb-21T09:12:29,"V3.90","W0JV","FOX24",144.225,"S6",8.63-V,34-mA,8.36-V,116-mA,-5Hrs
fox_clock.c    Todays_Loads/ 863 "FOX24": 2025-Feb-21T15:17:09,"V3.90","W0JV","FOX24",144.225,"S6",8.69-V,34-mA,8.39-V,115-mA,-5Hrs
fox_clock.c    Todays_Loads/ 863 "FOX25": 2025-Feb-21T09:24:09,"V3.90","W0JV","FOX25",144.225,"S6",7.79-V,41-mA,7.17-V,141-mA,-5Hrs,***Battery***
fox_clock.c    Todays_Loads/ 863 "FOX25": 2025-Feb-21T15:28:41,"V3.90","W0JV","FOX25",144.225,"S6",7.87-V,41-mA,7.22-V,140-mA,-5Hrs
fox_clock.c    Todays_Loads/ 863 "FOX26": 2025-Feb-21T09:06:23,"V3.90","W0JV","FOX26",144.225,"S6",8.73-V,29-mA,8.45-V,115-mA,-5Hrs
fox_clock.c           main/1402 CMD: rm sorted_foxlog.csv
fox_clock.c           main/1413 UPDATE Labels: fox_label.csv
fox_clock.c     Last_Write/1050 CMD: mv fox_label.csv fox_label.csv-2025_Feb_21T16:08:40
```

This will update the *fox_label.csv* file with the data on the last line of the *foxlog.csv* file, which may be redundant. It will nonetheless produce a report of the updates performed that day (earlier detail lines are ignored).

Any units with a low battery voltage (like **FOX25** which is below 7.2V when transmitting in this example) are noted in the report. This gives you the opportunity to replace the battery before the hunt so you don't drop a transmitter due to an exhausted battery.

13.6 Time from GPS NMEA and PPS

The 102-73181-10 board could, in theory, accept a NMEA sentence stream from a GPS for setting time. In practice, however, there is no practical means of getting the PPS signal into the Fox Transmitter.

Without the PPS signal and assuming the GPS is configured such that the NMEA sentence strings are delivered with a consistent offset from the start of a second (i.e. always in the same position, relative to the PPS signal) it should be possible to load the TOY clock.

Alas, this doesn't appear to be the case with the GPS18 used by the author. The GPS was configured to emit only the $GPGSA, $GPGSA and $GPGSA sentences.

Chapter 14

zNEO Programming Hardware and Utility

This utility is used to load the software image into the zNEO in the fox transmitter.

This utility is written in plain-old C to run on a Linux box. It makes use of the 102-73220-23 and 102-73220-33 boards to provide the programming connection into the target system.

This utility came about to address an issue with using the ZiLOG ethernet smart cable programmer. Something occurred with an update to Fedora40 that broke the **ZENETSC0100ZACG** device used by the author.

The replacement uses a standard *FTDIchip* USB UART device along with a small interface board to access the debug/programming port on the zNEO.

Help output from programming software:

```
./zNEO_P (V0.0/V2.10)
        ZiLOG ZNEO Programmer

    The following command line switches are recgonized:

    -h              help file, abbreviated
    -H              help file, include config file and device file list

    -r              Reset Target (and exit)
    -e              erase only (and exit)
    -p              skip erase
    -f <freq>       define target crystal frequency (default: 20.000 MHz)

    -d              scan target flash to _pre.hex onnly
    -1              scan target flash to _pre.hex  BEFORE programming
    -2              scan target flash to _post.hex AFTER  programming

    -x <file>       input hex file (filetype, not supplied, must be .hex)
    -X <file.hex>   reformatted hex file (you must supplt the .hex suffix!)
    -V              Verify only
    -l <file>       append to log file (include timetage on each line)

    -S <nickname>   Select device by name
                        EZ3PGM    38400
                        EZ5PGM    57600
                        EZ1PGM   115200
                        EZ2PGM   230400
                        EZ4PGM   460800
                        EZ8PGM   500000
```

Typical use:

```
./zNEO_P -SEZ5PGM -xfox_73181
```

The serial device selection is through the **-SEZ5PGM** argument. Note that this particular device operates the 57,600 bit/second rate to stay well within what the zNEO will tolerate. All of the listed bit rates should be within the capability of the zNEO *On Chip Debugger*. In practice, operation above 115,200 may not allow enough time for flash memory programming inside the zNEO. The **ZENETSC0100** programmer that this hardware replaces was operating at around 125,000 bits/second.

The basic processing time for programming a full (128K) device (erase, program, and verify), is a bit over 90 seconds. Page size is limited to 32 bytes during programming and expanded to 256 bytes during verify. Adding the **-1** adds about 30 seconds for the pre-read operation.

The hex file selection is through the **-xfox__73181** argument. The filetype of **.hex** is forced (the .hex must not appear). The device memory image can be saved both before and after programming using the **-1** and **-2** flags.

The hex file decoder in the utility is white-space insensitive. It will accept expanded hex files that have white-space embedded within the record as well as text appended to each line. The output produced when using the **-1** and **-2** flags will be correctly processed.

The pre and post **.hex** files are named based on the **-x** argument. The input filename is simjply the supplied argument with the **.hex** appended.

The **-1** will save a file collected before the device is erased. It is named by appending **__before.hex** to the supplied filename.

The **-2** will save a file collected after the device is programmed. It is named by appending **__after.hex** to the supplied filename. Note that this image is collected from the target after programming to verify that the device was successfully programmed.

The **-X** is used to produce an *annotated* hex file. This simply spreads out the hex record to make manual inspection a bit easier. There is also a text dump appended to the hex data.

The **ZiLOG ZNEO Programmer** software will directly deal with this output. It removes whitespace as the line of text is read. Extra characters after the checksum field are ignored.

14.1 Single Channel UART

This is the base board that holds the USB UART.

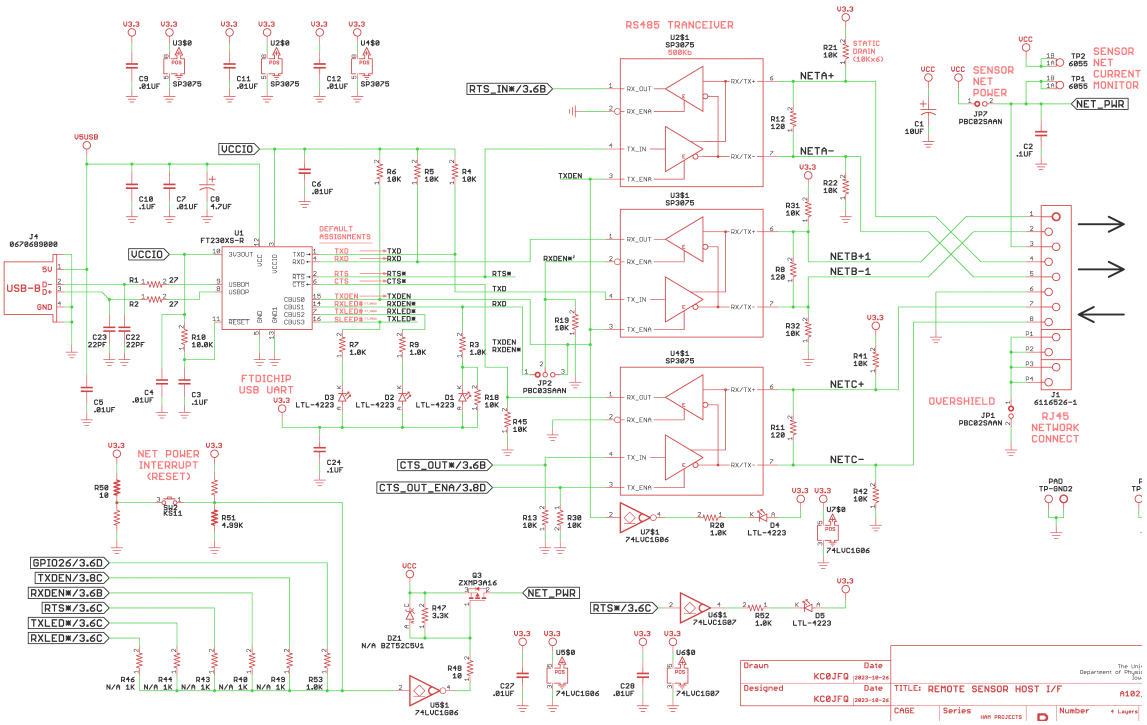


Figure 14.1: Single Channel UART

U1 is the USB UART device that provides a connection to the host system.

For our programming application the RS485 interface U2..U4 is not populated. We use this board as a place for the FT230X to live and a mounting point for the programming board.

The circuit provides visual indicators (LEDs) to allow monitoring activity.

This is the base board connector to the programming board.

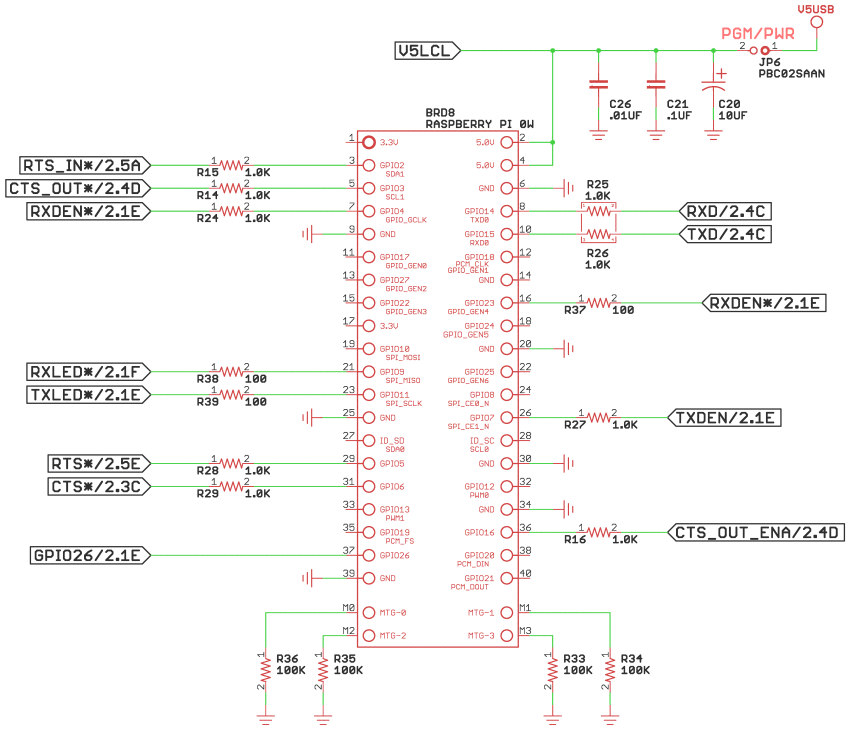


Figure 14.2: base board to programming board

This board was originally designed as a RS485 interface for a sensor network. In addition to the USB interface the board will also accommodate a *Raspberry PI ZERO* in place of the USB UART.

The *Raspberry PI ZERO* connector provides the interface to the programming adaptor board. The R25/R26 resistor pair provides for swapping the TxD and RxD lines to the *Raspberry PI* connector should that be necessary.

14.2 ZiLOG eZ8 Programming Adapter

This is the full-duplex (FTDIchip device) to half-duplex (zNEO) interface drawing:

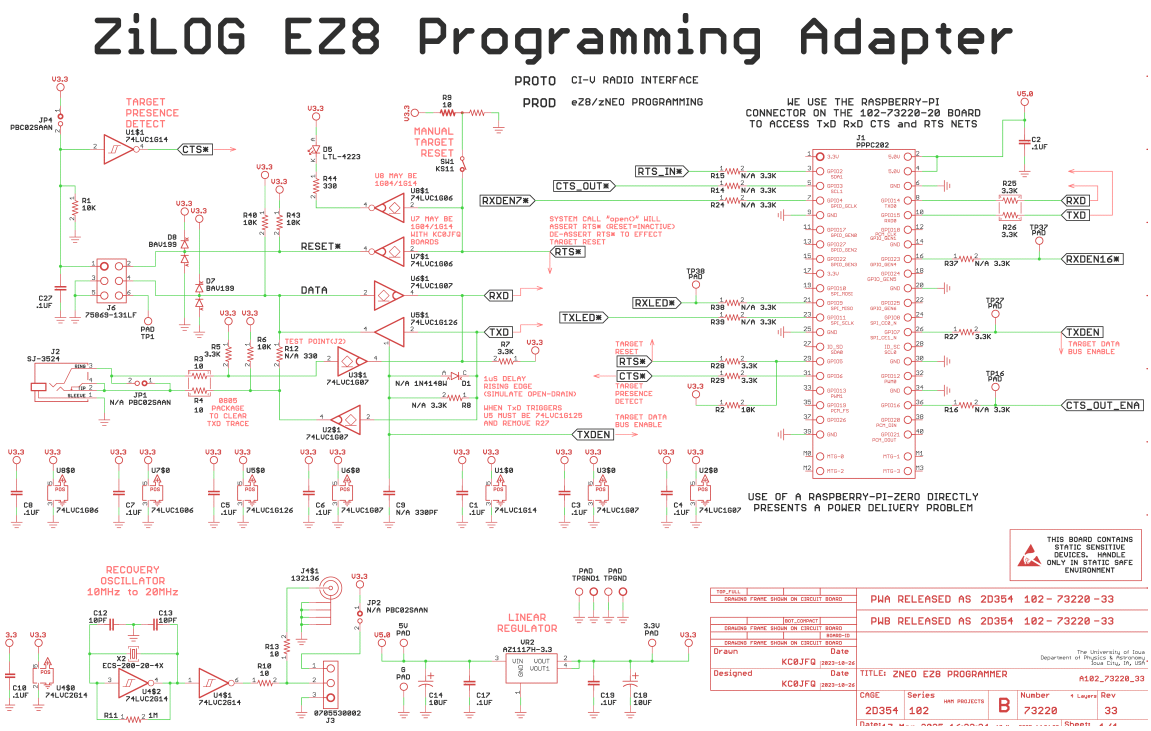


Figure 14.3: eZ8 Adapter

The programming board provides the interface between the full-duplex UART (TxD and RxD data) and the half-duplex interface to the eZ8/zNEO target. The eZ8/zNEO interface is open drain to allow either side to drive the data bus.

To improve speed, data from host(J1) to target(J6) is buffered using a tri-state gate(U5) that is enabled when transmitting. The use of a push-pull gate improves rise time for data being sent to the target.

The 74LVC1G126 device is pin compatible with a 74LVC1G07 open drain driver should this prove useful. One would epect to control the enable pin of U5 using the **TXDEN** net as the FT230X will set the TXDEN bit when it transmits and clear the bit when not transmitting.

The **Recovery Oscillator** shown in the lower left corner is there to provide a clock to a board where the zNEO internal oscillator configurtion has been corrupted or improperly programmed. Pin 2 can be used by itself to effect the recovery operation by connecting to the XIN pin on the zNEO (LQFP64 pin 64). Power and ground appear on the connector (J3) should it become necessary for either of these signals to be used.

It should be possible to use a *Raspberry PI ZERO* to drive this board. The R25/R26 resistor pair is also present on this board to allow TxD and RxD to be correctly routes.

The 102-73220-33 board is not mechanically well suited for use with the *Raspberry PI ZERO* as it extends outside the edges of the *Raspberry PI ZERO*. It should, none-the-less be electrically compatible.

Power to the *Raspberry PI ZERO* may present a problem. A regulated 5V supply would need to be connected to either the *Raspberry PI ZERO* has a microUSB conenctor for providing power which may be used to also power the programming board. There are also a pair of pads on the 5V and GND nets on the programming board that may be used to feed the *Raspberry PI ZERO*.

14.3 Four Channel UART

This is a proposed UART motherboard with 4 identical channels that fits in our Hammond 1599E box. This board is intended to condense the connections required to debug the Fox Transmitter to a single board.

One channel to connect to the zNEO UART-0, a second channel to connect to the zNEO UART-1, and the third channel to program the zNEO,.

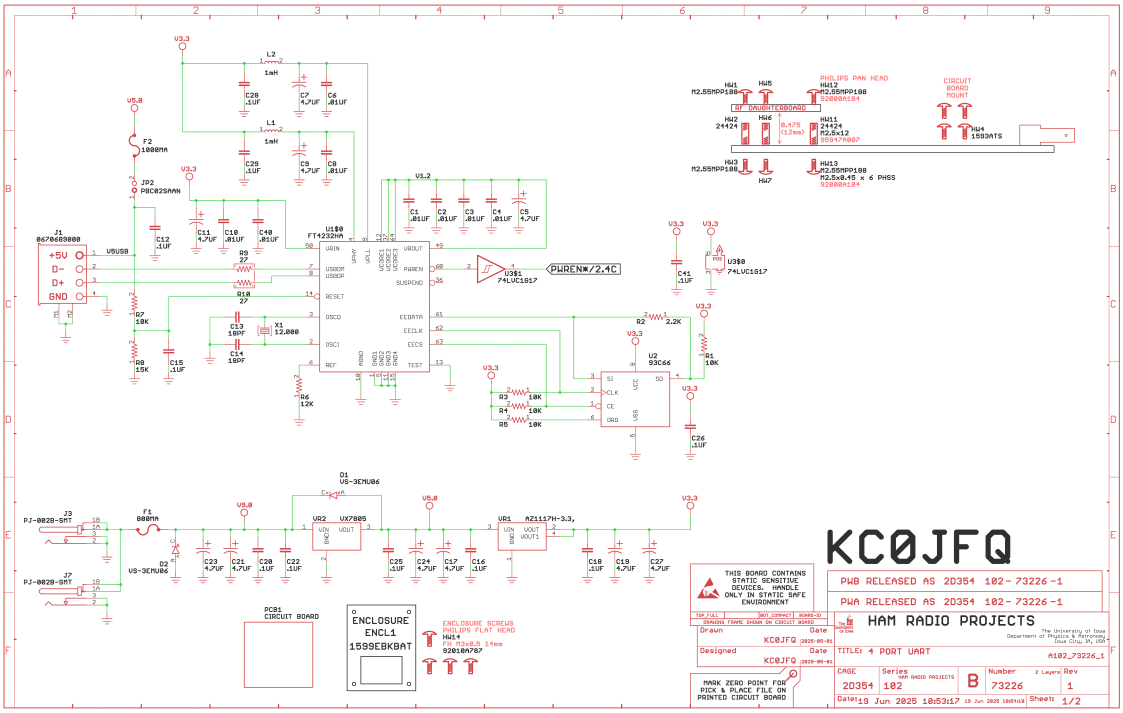


Figure 14.4: FTDIchip FT4232

This makes use of the **FTDIchip** *FT4232* quad UART to provide 4 serial channels in a single 64 ping flat-pack.

The motherboard has 4 plug-in positions for daughtercards to provide the individual electrical interface.

Any combination of daughterboards may be installed on the motherboard.

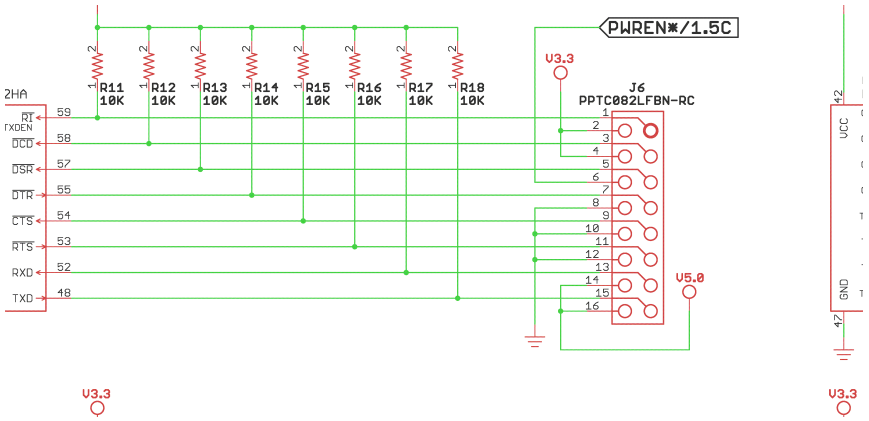


Figure 14.5: FTDIchip FT4232 channel

Each of the other three channels are wired identical to the first.

Shown here is the connections from the *FT4232* to the daughtercard.

14.3.1 UART 3.5mm Channel Card

This is the logic-level UART interface used by the Fox Transmitter.

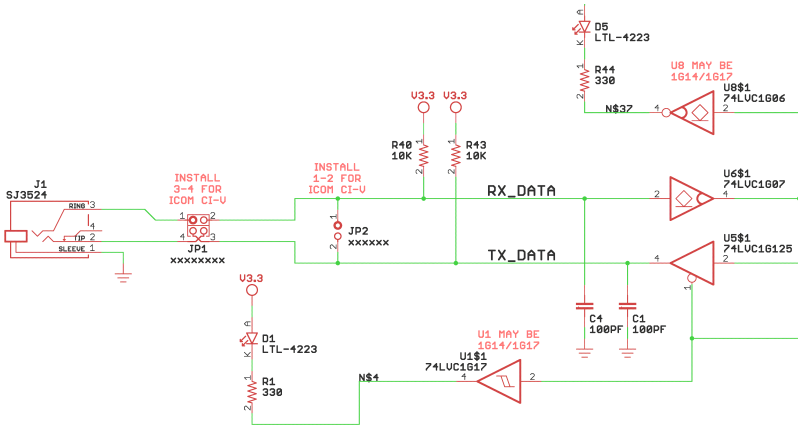


Figure 14.6: 3.5mm serial channel

This is used to connect to the primary 3.5mm connector on the Fox Transmitter to load FRAM and FLASH. It is also used to connect to the other serial channel through the 102-73181-24 daughtercard.

14.3.2 UART 3.5mm Isolated Channel Card

This logic-level UART interface is also available with isolation.

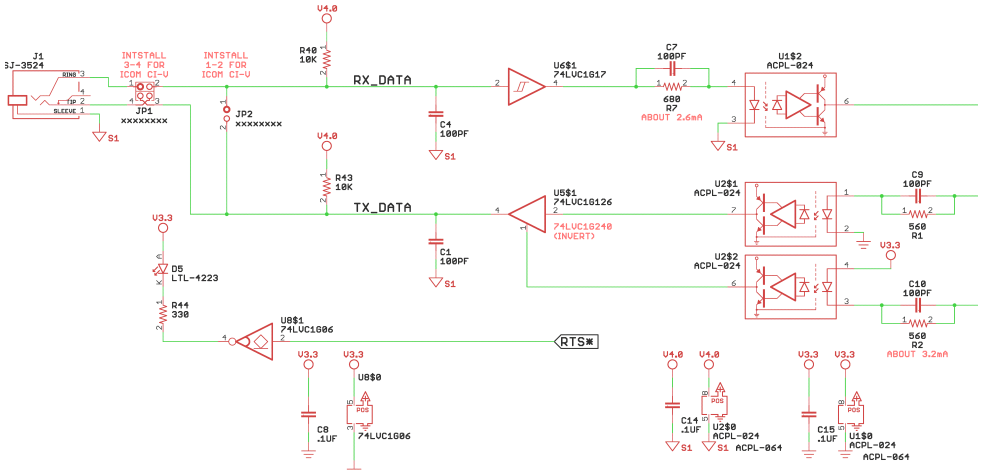


Figure 14.7: 3.5mm serial channel, isolated

This provides electrical isolation between the host computer (through the USB cable) and the Fox Transmitter.

Optical isolators are inserted between the motherboard UART and the level shifters. The indicated opto-isolator device has very low drive requirements to reduce the current load presented by the LED in U1. The ACPL-024 is a high speed isolator that should have no difficulty running at the 115,200 b/S rate used for fast loading the FRAM/FLASH memory.

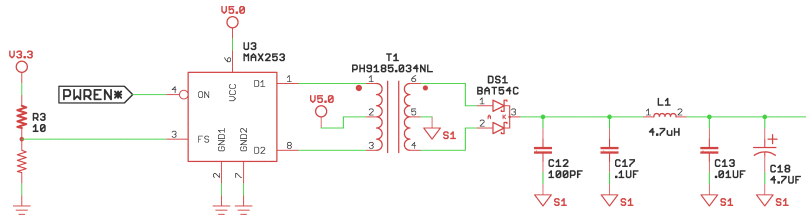


Figure 14.8: Isolated 5V supply

The isolated power is supplied using a MAX253 driver and a PH9185.034NL isolation transformer. This power convertor runs off of the 5V rail (it has higher current capability than the 3.3V rail). The power convertor stops running when the FT4232 is in **shutdown** (follow the **PWREN*** net).

14.3.3 ZiLOG eZ8 Programming Channel Card

This simply duplicates the programming channel from the eZ8 Programming Adapter in section 14.2 on page 261.

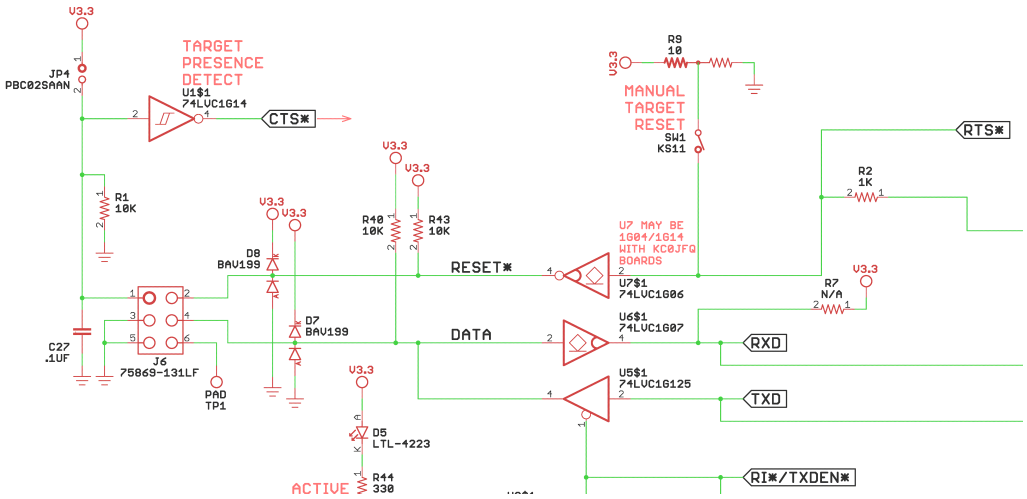


Figure 14.9: eZ8 Programming Card

The *recovery oscillator* (section 14.2) is removed owing to space constraints.

This interface should behave identically to the eZ8 Programmer above.

14.4 FTDIchip EEPROM programming

All of the sub-projects presented here make use of FTDIchip USB-UARTS. The EEPROM typically will need to be loaded with configuration information spefific to the application where the chip is used.

We will first use the *ftdi_eeprom* utility under Linux to extract the current data in the EEPROM, saving it to a *.conf* file. This *.conf* file can then be modified to suit your needs.

We can then use the *ftdi_eeprom* utility under Linux to program the EEPROM in the FTDI UART with new data.

Vendor_ID and Product_ID

Always use the values provided in the device before it is programmed. We use the standard serial drivers so there is no good reason to change these values.

Serial Numbers

Serial numbers **must** be unique. There is a numbering convention used by the author to serialise all FTDI devices. In the examples below, the last 4 characters of the serial number are sequential across all devices produced by the author. The initial characters serve to identify the type of device and are not detailed here.

Max Power

All of these device should fit comfortably within the power constraints imposed by the USB standard.

You may consult the datasheets to estimate the maximum power that should be required by the various projects.

14.4.1 EEPROM, commands

The command used by the author to read the EEPROM in the USB-UART device.

```
#!/usr/bin/bash
sudo \
    ftdi_eeprom \
        --verbose \
        --device i:0x0403:0x6001 \
        --read-eeprom \
        $1.conf
```

The command used by the author to write the EEPROM in the USB-UART device.

```
#!/usr/bin/bash
sudo \
    ftdi_eeprom \
        --verbose \
        --device i:0x0403:0x6001 \
        --flash-eeprom \
        $1.conf
```

Configurable string values.

Decode Macro	Key Name	Description
CFG_STR	manufacturer	String we provide
CFG_STR	product	String we provide
CFG_STR	serial	String we provide (unique!)
CFG_STR	user_data_file	
CFG_STR	filename	

Configurable integer values.

Decode Macro	Key Name	Description
CFG_INT	vendor_id	**DO NOT ALTER**
CFG_INT	product_id	**DO NOT ALTER**
CFG_INT	default_pid	**DO NOT ALTER**
CFG_INT	max_power	calculate this value
CFG_INT	eeprom_type	
CFG_INT	release_number	
CFG_INT	usb_version	**DO NOT ALTER**
CFG_INT	user_data_addr	

Most of the devices supplied by FTDIchip have pins that can be configured by the EEPROM to route useful signals to these pins.

Consult the datasheet to determine which of the *cbus* groups are supported by the device. You can then configure them as needed. Reading the existing EEPROM usually will give you clues as to which *cbus* group to choose from.

In the way of example, the FT230**X** device lists only 4 cbus pins, so pick from the *cbus**x*** group. The FT232R has 5 *cbus* pins so here we use the *cbus* group. And finally, the FT232H, with 10 *cbus* pins, will use the *cbus**h*** group.

Alternate features for legacy FT232R devices (USB port on the Fox Transmitter).

Decode Macro	**Key Name**	**Description**
CFG_INT_CB	cbus0	
CFG_INT_CB	cbus1	
CFG_INT_CB	cbus2	
CFG_INT_CB	cbus3	
CFG_INT_CB	cbus4	

CBUS Options	**Description**
TXDEN	Output Driver Enable (RS485)
PWREN	USB configured, USB suspend: high
TXLED	TX activity status LED
RXLED	RX activity status LED
TXRXLED	TX & RX activity status LED
SLEEP	USB suspend
CLK48	48 MHz clock
CLK24	24 MHz clock
CLK12	12 MHz clock
CLK6	6 MHz clock
IOMODE	IO port for CBUS bit bang mode
BB_WR	Synchronous Bit Bang Write strobe (FT232R and FT-X only)
BB_RD	Synchronous Bit Bang Read strobe (FT232R and FT-X only)

Alternate features for FT232H/FT2232H/FT4232H devices (used on the 102-73226-B boards).

Decode Macro	**Key Name**	**Description**
CFG_INT_CB	cbush0	
CFG_INT_CB	cbush1	
CFG_INT_CB	cbush2	
CFG_INT_CB	cbush3	
CFG_INT_CB	cbush4	
CFG_INT_CB	cbush5	
CFG_INT_CB	cbush6	
CFG_INT_CB	cbush7	
CFG_INT_CB	cbush8	
CFG_INT_CB	cbush9	

CBUS-H Options	Description
TRISTATE	IO Pad is tri-stated
TXLED	TX activity status LED
RXLED	RX activity status LED
TXRXLED	TX & RX activity status LED
PWREN	USB configured, USB suspend: high
SLEEP	USB suspend
DRIVE_0	Drive constant 0 (FT232H and FT-X only)
DRIVE1	Drive constant 1 (FT232H and FT-X only)
IOMODE	IO port for CBUS bit bang mode
TXDEN	Output Driver Enable (RS485)
CLK30	30 MHz clock
CLK15	15 MHz clock
CLK7_5	7.5 MHz clock

Alternate features for FT230X devices (used on the 102-73220-20 and 102-73220-23 boards).

Decode Macro	Key Name	Description
CFG_INT_CB	cbusx0	
CFG_INT_CB	cbusx1	
CFG_INT_CB	cbusx2	
CFG_INT_CB	cbusx3	

CBUS-X Options	Description
TRISTATE	IO Pad is tri-stated
TXLED	TX activity status LED
RXLED	RX activity status LED
TXRXLED	TX & RX activity status LED
PWREN	USB configured, USB suspend: high
SLEEP	USB suspend
DRIVE_0	Drive constant 0 (FT232H and FT-X only)
DRIVE1	Drive constant 1 (FT232H and FT-X only)
IOMODE	IO port for CBUS bit bang mode
TXDEN	Output Driver Enable (RS485)
CLK24	24 MHz clock
CLK12	12 MHz clock
CLK6	6 MHz clock
BAT_DETECT	Battery Charger Detect (FT-X only)
BAT_DETECT_NEG	Inverse signal of BAT_DETECT (FT-X only)
I2C_TXE	Transmit buffer empty (FT-X only)
I2C_RXF	Receive buffer full (FT-X only)
VBUS_SENSE	Detect when VBUS is present via the appropriate AC IO pad (FT-X only)
BB_WR	Synchronous Bit Bang Write strobe (FT232R and FT-X only)
BB_RD	Synchronous Bit Bang Read strobe (FT232R and FT-X only)
TIME_STAMP	Toggle signal each time a USB SOF is received (FT-X only)
AWAKE	Do not suspend when unplugged/disconnect/suspsend (FT-X only)

Configure channel type.

Decode Macro	Key Name	Description
CFG_INT_CB	cha_type	
CFG_INT_CB	chb_type	

Channel Type	Description
UART	
FIFO	
OPTO	
CPU	
FT1284	

Configure drive strength.

Decode Macro	Key Name	Description
CFG_INT_CB	group0_drive	

Group-0 Drive	Description
4MA	
8MA	
12MA	
16MA	

Configurable Feature Enable Flags.

Decode Macro	Key Name	Description
CFG_BOOL	change_usb_version	
CFG_BOOL	flash_raw	
CFG_BOOL	high_current	
CFG_BOOL	in_is_isochronous	
CFG_BOOL	out_is_isochronous	
CFG_BOOL	remote_wakeup	
CFG_BOOL	self_powered	*true* powered external to USB *false* powered by USB
CFG_BOOL	suspend_pull_downs	
CFG_BOOL	use_serial	*true* to use the serial number string

Configurable Feature Enable Flags.

Decode Macro	Key Name	Description
CFG_BOOL	cha_rs485	**TRUE** for TXDEN*
CFG_BOOL	chb_rs485	**TRUE** for TXDEN*
CFG_BOOL	chc_rs485	**TRUE** for TXDEN*
CFG_BOOL	chd_rs485	**TRUE** for TXDEN*
CFG_BOOL	cha_vcp	**TRUE** for *virtual com-port driver*
CFG_BOOL	chb_vcp	**TRUE** for *virtual com-port driver*
CFG_BOOL	chc_vcp	**TRUE** for *virtual com-port driver*
CFG_BOOL	chd_vcp	**TRUE** for *virtual com-port driver*

Configurable pin polarity flags.

Decode Macro	Key Name	Description
CFG_BOOL	invert_cts	
CFG_BOOL	invert_dcd	
CFG_BOOL	invert_dsr	
CFG_BOOL	invert_dtr	
CFG_BOOL	invert_ri	
CFG_BOOL	invert_rts	
CFG_BOOL	invert_rxd	
CFG_BOOL	invert_txd	

14.4.2 EEPROM, FOX17.conf

Configuration for the FOX17 Fox Transmitter. This is 102-73181-0 hardware where the only command path is through the on-board FT232 UART.

```
vendor_id=0x403
product_id=0x6001
max_power=24
manufacturer="KC0JFQ"
product="KC0JFQ_FOX_V3"
serial="2078-0-0114"
filename="FOX17.bin"
self_powered=false
use_serial=true
```

The *self_powered=false* line indicates that the device is accepting 5V power from the USB bus. The USB-UART on the Fox Transmitter is powered through the USB bus so it remains active when the Fox Transmitter is not powered. This keeps the USB devices from dropping off the bus when powwer is removed.

14.4.3 EEPROM, EZ8PGM.conf

Configuration for the eZ8 programmer.

```
vendor_id=0x403
product_id=0x6015
max_power=400
manufacturer="KC0JFQ"
product="EZ8PGM"
serial="42B3-0-0118"
filename="EZ8PGM.bin"
self_powered=false
use_serial=true
#
cbusx0="TXDEN"
cbusx1="IOMODE"
cbusx2="RXLED"
cbusx3="TXLED"
```

The eZ8 programmer is intended to obtain power from the USB bus as the power requirements for the programmer are minimal.

We do advertise a higher power level than we actually use.

14.4.4 EEPROM, prototype

Configuration for the some other prototype.

```
vendor_id=0x403
product_id=??
max_power=??
manufacturer="KC0JFQ"
product="produce name"
serial="serial number"
filename="prototype.bin"
self_powered=false
use_serial=true
```

Set the *self_powered=* line as needed.
Set the *serial=* line to a unique value so it can be cleanly discovered and not conflict.

14.4.5 EEPROM, radio 25.conf

Configuration for the KC0JFQ radio interface. This project uses the FT4232 device.

```
filename=radio_25.bin
vendor_id=0x0403
product_id=0x6011
eeprom_type=0x66
manufacturer="Perf Proc"
product="KC0JFQ Radio I/F"
serial="2078-0-0025"
use_serial=true
max_power=0
self_powered=true
remote_wakeup=false
cha_type=UART
chb_type=UART
cha_rs485=false
chb_rs485=false
chc_rs485=false
chd_rs485=false
cha_vcp=true
chb_vcp=true
chc_vcp=true
chd_vcp=true
```

The USB-UART is powered externally (not by the USB bus) so the *serial=* reflects thi use.

14.4.6 EEPROM,

Configuration for the .

```
ver
```

Chapter 15

Audio File Utility

The audio utility program, **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*.

15.1 Input File

The input file is a standard WAV file.

It must be PCM data, 8 bits wide, unsigned, at 4,000 or 5,000 samples/second.

15.2 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.

15.2.1 Audio Utility command line (typical)

This is the command used to render an audio fragment into a loadable hex file. This is repeated for each *utterance* that is to be loaded into the Fox Transmitter.

```
sox -v 0.90 voice_01.wav --rate 4000 --bits 8 R4K_voice_01.wav
mv R4K_voice_01.wav voice_01.r4k
audio_util -m(0,0) -b -a 0x0000 -H -i voice_01.r4k -o voice_01.hex
```

We then use a shell script to iterate over all the files needed by the Fox Transmitter.

In this fragment, the **FILE=voice_01** assignment is repeated for each audio clip (utterance).

```
SOX=/usr/bin/sox
MV=/usr/bin/mv
AUDIO=/home/wtr/Radio/halo_term/audio_util
ADDR="-m(0,0) -b -a 0x0000"
FILE=voice_01
```

Down in the loop, we update the **FILE** variable as we iterate over the list of files. The **SOX** utility resamples the input to our needed 4KHz 8-bit mono file. We then rename ($MV) the file to differentiate the .wav files from the resampled files.

The, as we run the **audio utility**, we save the report (that is produced at the end) that tells us where to start the next audio clip.

```
$SOX -v $VOLU $FILE.wav --rate 4000 --bits 8 R4K_$FILE.wav
$MV R4K_$FILE.wav $FILE.r4k
ADDR=$($AUDIO $ADDR  -H   -i $FILE.r4k -o $FILE.hex)
```

As we cycle through the list of files, updating the **FILE** variable, we produce a group of hex files with each having a load address that follows the previous file.

At the end, we concatenate all the .hex files into one to be loaded using the **fox_simple** utility.

Embedded in the master .hex file are the commands needed to form the *TALK Directory*. We use **grep** to extract them and remove the leading *REM-* command. This results in a *TALK* record for each utterance in the master .hex file.

Although we could leave the *TALK Directory* embedded in the .hex file, the talk directory needs to be loaded following an erase of the FRAM. The FLASH, once loaded, we prefer to simply leave it alone since it takes so long to load.

15.2.2 Audio Utility command line arguments

-H dump RIFF/WAV header

This flag causes a dump of the RIFF/WAVE header to be included in the .hex file.
The **fox_simple** utility and the Fox Transmitter ignore this content. This means, of course, it is there for your inspection.

-i <input file>

```
input file name
defaults to stdin
use this name for output with ".hex"
unless "-o" overrides
```

This should be obvious.
See the above example.

-o <output file>

```
output file name
defaults to stdout or input.hex
```

This should be obvious.
See the above example.

-a <address> in hexadecimal!

```
starting address in hexfile
defaults to 0x0000
```

This argument, the starting address of the hex data, will be unique for each file that makes up the master file.

Each utterance is stacked, one after the other, in the FLASH memory.

-d <count> in decimal

```
strip 'count' header bytes
defaults to zero
```

Header delete count.
All version of the Fox Transmitter software you will encounter knows about the RIFF/WAVE header in a wav file. The Fox Transmitter software obtains output parameters from the RIFF/WAVE header.

This allowed for removing the RIFF/WAVE header for early version of the Fox Transmitter Software.

-D <count> in decimal

```
offset 'count' header bytes
defaults to zero (leaves RIFF header intact)
```

This is part of the same accommodation from above.
This offsets the address in the TALK Directory entry by the supplied value.

-s <sample rate>

```
audio sampling rate
defaults to "4K"
```

This is more of the accommodation from above.
When a RIFF/WAVE header is not present, the sample rate needs to appear in the TALK Directory entry. This provides the means to supply that for non standard files.

-b "batch mode"

```
suppress all output other than
the address of the next file
```

Suppress all reporting.
The only output from the program are properly formatted command line flags and arguments. This, of course, enables batch processing in a shell script (see example above).

-m <(min,max)>

```
emit min max values found in file
```

The gives us an indication of the volume of the utterance. Since this is an 8-bit unsigned sample stream, minimum value is 0 and the maximum is 255.

-c HEX File cluster size (128)

```
MUST be even power of 2!
```

The parameter determines the start address that will be delivered to the next file. Typical value is 32 to align with the record size of the generated .hex file. This makes it easier to view the header data.

15.3 Downloading the audio image

Downloading into the Fox Transmitter may be accomplished using either the **fox_simple** utility or the **fox_binary** utility. The audio file we will download is plain text (in the form of InTel HEX Records) that can bed fed to the command decoder in the Fox Transmitter through the control port (i.e. the 3.5mm jack). The **fox_simple** utility streamlines the process of feeding command lists and hex files to the target Fox Transmitter.

We can signifigantly speed up the process by using the **fox_binary** utility (the preferred method). The **fox_binary** utility takes the same InTel HEX file and triggers the binary loader in the Fox Transmitter.

The **fox_binary** utility does **not** share! You must skill off any copies of **halo_term** that is monitoring the Fox Transmitter command port. If **halo_term** is left running, the **fox_binary** utility will switch the Fox Transmitter into binary mode but handshake characters will be consumed by **halo_term** which will hang things up.

An effective way to avoid interference is to run **fox_binary** and **halo_term** from one terminal session. The **fox_binary** utility will cause the Fox Transmitter to revert to accepting commands at the standard bit rate and then exit. On the other hand, **halo_term** must be aborted manually (use control-C).

See the desctiption of using the **fox_binary** utility in section 12.4 on page 239.

Progress of the **fox_simple** utility 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 \
        -SFOX2X \
        -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 downloaded.

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.

Also keep in mind that the HEX file will have directory entries embedded. These directory entries will not be saved as they are prefixed with a *Remark* command stem.

Example Audio File Fragments:

```
REM-  Input File: V_FOX34.r4k
REM- Output File: V_FOX34.hex
REM-  RIFF/WAVE HEADER DUMP Input File: V_FOX34.r4k
REM- ChunkID         RIFF
REM- ChunkSize       5616
REM- Format          WAVE
REM- Subchunk1 ID    fmt
REM- Subchunk1 Size  16
REM- Audio Format    1
REM- Num Channels    1
REM- Sample Rate     4000
REM- Byte Rate       4000
REM- Block Align     1
REM- Bits per Sample 8
REM- Subchunk2 ID    data
REM- Subchunk 2 Size 5580
REM- esav TALK=V_FOX34 264960
:02 0000 04 0004 F6
:20 0B00 00 52494646F015000057415645666D74201000000001000100A00F0000A00F0000 9F
:20 0B20 00 0100080064617461CC150000807F808081808080807F8081828218080808180 2B
:20 0B40 00 7F8181808080807F7F7F7F8081808080807F7F807F818081808080808080808080 98
   .
   .
   .
:20 20A0 00 7F8080808080807F8080818181818080818080818182818181808180818182 11
:20 20C0 00 81808080817F818181817F80807F7F7F8081808081817F808080818180808080 FB
:20 20E0 00 8181808181818080808080808081818181807F807F7F8081810000000000000000 D8
:00 0000 01 FF
REM- FLASH Allocation Range:  0x040B00-0x0420FF
REM- FLASH Cluster  MASK:FFFFFF80  Increment:127
REM- Next Step Arguments:  -b -a 0x42100 -m(0,240)
REM- Audio min/max:   1(    6)  240(    2)         V_FOX34.hex
REM- * * * * * * * * * * * * * * * * * * * *
```

The **audio_utility** program generates an annotated HEX file for loading into the **fox transmitter** system.

The header lines provide the *RIFF/WAVE HEADER DUMP* that decode the RIFF/WAVE waveform file. The **fox transmitter** used the *Sample Rate* and *ChunkSize* fields directly. The *Num Channels* and *Bits per Sample* fields are checked to verify mono 8 bit samples.

There are 3 type of *InTel HEX records* generated by the **audio_utility** program: A **linear address** record, a **data** record, and an **end of file** record.

The **linear address** records appear at the begining of each chunk and every 16K samples in the HEX file (whenever the lower 12 bits of the record address are zero).

The **audio_utility** program adds some follow-up notes at the end of the chunk.

The **FLASH Allocation Range** simply indicates the range of addresses in FLASH memory occupied by this chunk of audio data.

The **FLASH Cluster** line shows the cluster allocation control field. Each new chunk starts on a boundary controlled by the *MASK* field. In this example, a new chunk starts on a 128 bye boundary.

The *Next Step Arguments* show the control arguments that will be passed along to the next invocation of the **audio_utility** program. Note that the **-a** field (the starting address) is after the end of the **FLASH Allocation Range**.

The *Audio min/max* shows counts of the number of samples at minimum volume and maximum volume.

15.3.1 Downloading at 115,200 b/S

Use the **fox_binary** utility to perform a high speed load. A description may be found in section 12.4 on page 239.

The **fox_binary** utility will send one of the following commands:

- **H56K PROG**
- **H115 PROG**
- **H56K WAVE**
- **H115 WAVE**

This switched bit rates and selects either the FRAM (PROG) or the FLASH (WAVE) and switches to the binary protocol.

15.4 SOX

We use **sox** to resample audio data down to the require 4KHz or 5KHz sample rate. The sample size is also reduced to mono 8 bits.

A typical invocation:

```
sox -v1.0 \$FILE.wav --rate 4000 --bits 8 R4K\_\$FILE.wav
sox -v1.0 \$FILE.wav --rate 5000 --bits 8 R5K\_\$FILE.wav
```

Volume is adjusted with the **-v** flag and the filename, as you would expect, is provided through the $FILE variable.

The –**rate** *4000* selects the target sample rate. Shown are selections for 4KHz and 5KHz. The **fox transmitter** will deal with 4KHz, 5KHz, 8KHz, 10KHz and 16KHz. Note that 8KHz and above well exceeds the bandwidth allowed on the 2 meter band so should not be used.

The –**bits** *8* selects the sample width. This being required to be 8 bits.

I am not sure why we don't specify single channel with –**channels** *1*. It should not cause problems to explicitly specify this.

We should look at –**encoding** to see if we can improve audio quality using *a-law* or *u-law*.

`https://sourceforge.net/projects/sox/`

15.5 Audacity

Audacity is used to edit voice clips for use in the Fox Trasmitter. Any example voice clips provided with the transmitter were edited using the **Audacity** utility.

`https://www.audacityteam.org/`

15.6 cwwav

The **cwwav** utility can be used to generate waveforms of *Morse code* data. The density of code traffic in a wav file is far less than that achieved using the Fox Transmitter code generator. The **cwwav** utility does provide for envelope shaping that the inernal code generator will not produce for you.

A file produced by **cwwav** obtained from github needs to be resampled using **sox** to rework the wav file to 8-bit unsigned mono data sampled at 4KHz. A modified **cwwav** utility is used by the author to directly produce *.wav* files than can be processed by the

`https://github.com/oyvholm/cwwav`

Chapter 16

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.

16.1 fox_label program

This is the utility that produces all the printable documents for the hunt. Running it produces the following:

Table 16.1: fox_label output files

	Type	**Contents**	**Description**
1	label	fox_label_A_bot.ps	Transmitter FCC Identification
2	label	fox_label_A_top.ps	Fox Hunt Identification
3	sheet	fox_label_A_chk.ps	Fox Hunt Setup Checklist
4	card	fox_label_B_cards.ps	Fox Hunt Participant Log Card
5	card	fox_label_B_quick_cards.ps	Fox Hunt Participant Log Card
6	card	fox_label_B_hunter.ps	Fox Hunt Participant Card
7	label	fox_label_B_power.ps	Fox PA power labels
8	card	fox_label_C_FOX0.ps	Fox Hunt Transmitter Found Card
n	card	fox_label_C_FOXn.ps	(13 more of them in)

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 could be used inadvertantly. Using the wrong starting offset (see -o on page 284) 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 16.9 on page 290.

First we generate the pages needed for the hunt. The club where these were developed can deploy up to 17 Fox Transmittters. We operate 12 stations in two groups, so it is only necessary to generate labels for 12 units.

```
/home/wtr/Fox_Transmitter/trunk/fox_label \
        -c "fox_label.csv" \
        -o 0 \
        -m 12 \
        -C W0JV \
        -e "ICARC Fox Hunt" \
        -A 5371
```

Now we can print the check sheet that has information for the primary transmitter set (i.e. the first 12 stations). The check sheet has 14 entries, so we get a couple of extras.

```
lp fox_label_A_chk.ps           <--- plain paper
ls -l fox_label_A_bot.ps        <--- we do these only once!
```

There are 5 stations that are used for training, so we don't bother with check sheets and labels for them.

Load 14-up stock for the transmitter labels that are updated for each hunt. These have unique serial numbers that will be recorded by the hunters.

```
lp fox_label_A_top.ps           <--- 14-up label stock
```

The hunter log cards aren't unique, print more if needed.

```
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).

```
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
```

These are the labels for the second hunt group. This hunt group operates on a different frequency.

```
lp fox_label_C_FOX27.ps          <--- 10-up business cards
lp fox_label_C_FOX28.ps          <--- 10-up business cards
lp fox_label_C_FOX29.ps          <--- 10-up business cards
lp fox_label_C_FOX30.ps          <--- 10-up business cards
lp fox_label_C_FOX31.ps          <--- 10-up business cards
lp fox_label_C_FOX32.ps          <--- 10-up business cards
```

The remaining transmitters are operating as training aids so the labels unique to the hunt aren't updated. The novice hunter can simply note on the hunt card that the transmitter was located.

16.1.1 fox_label -h

Help text.

16.1.2 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.

16.1.3 fox_label -C

SECONDARY_Callsign.

Sets the callsign that appears on labels, cards and log sheets.

16.1.4 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.

16.1.5 fox_label -m

Check_Page_Count.

Sets the number of entries on the check sheet.
This is used to limit the number of transmitters reported on the check sheet when processing more transmitters that will fit on that check sheet.
This avoids creating duplicate labels which would have un-matched serial numbers.

This is also useful in keeping unused transmitters off of the list. We don't have to fill the page out!

16.1.6 fox_label -d

DBG_Level.

Increases the debug level by one.
Currently level 1 is everything!

16.1.7 fox_label -e

Event Name.

Sets the event name that appears on labels, cards and log sheets.

16.1.8 fox_label -p

Contact Phone Number.

Argument sets the phone number that appearts on the *fox_label_A_bot.ps* labels.

16.1.9 fox_label -P

Photo background scale.

Argument may appear multiple times.

This provides a set of labels and a scale for photographing boards. Each additional **-P <arg>** adds a line of text.

The **-p <offset>** is not used to hold a phone number as the standard set of labels are not generated when **-P** is used. Rather this argument is an offset applied to the scale to move the X axis zero left or right.

16.2 fox_label_A_bot

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.

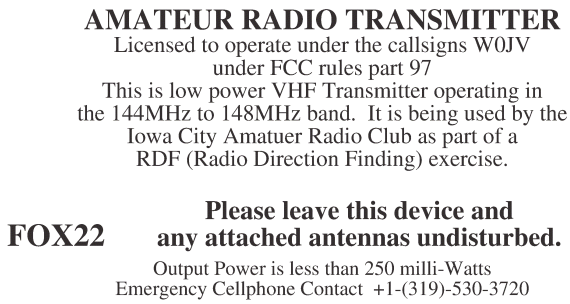


Figure 16.1: FOX Transmitter BOT Labels

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 to when the fox transmitter is first put into service. This helps prevent the identity of the transmitter being switched with another unit when the enclosures are opened.

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.

16.3 FOX HUNT Checkin Card

The *Hunter Cards* can be printed as a convenient means of keeping track of the participants.

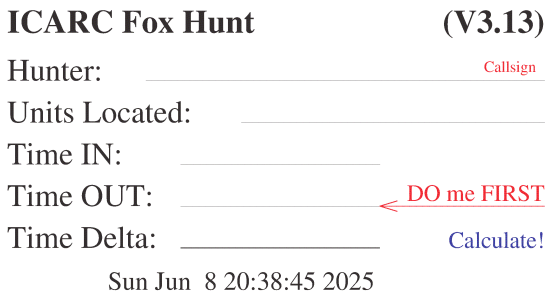


Figure 16.2: FOX HUNT Checkin Card

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.

These can stay at the hunt operators position or they may travel with the hunters.

16.4 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_top** file must be printed from the same run in order for the serial numbers to match!

ICARC Fox Hunt

Transmitter Time Synchronization Checklist (V3.13)

Sun Jun 8 20:38:45 2025 fox_label_A_chk.ps

Event Callsign: **W0JV** Start: End:

Name	Chk Freq Oper Freq	Drwg Nmbr Daughterboard	Power S/W Ver	Batt Voltage Run Schedule	Batt Current	Event Validation Code Event ID Code	FRAM FLASH
FOX21 Unit: 1	144.150MHz 144.225MHz	102_73181_10 102_73181_28	70mW V4.08	8.00V Idle Tx **7.60V** S6	24mA Idle Tx 122mA	144.225 / **5ISOYQR9** Off:1 ID: **093872**	256K 4096K
FOX22 Unit: 2	144.150MHz 144.225MHz	102_73181_10 102_73181_28	80mW V4.08	8.90V Idle Tx **8.60V** S6	38mA Idle Tx 123mA	144.225 / **BBTEST4Q** Off:20 ID: **013726**	256K 8192K
FOX23 Unit: 3	144.150MHz 144.225MHz	102_73181_10 102_73181_28	95mW V4.08	7.90V Idle Tx **7.20V** S6	54mA Idle Tx 157mA	144.225 / **CJ0712P9** Off:13 ID: **827718**	256K 8192K
FOX24 Unit: 4	144.150MHz 144.225MHz	102_73181_10 102_73181_28	60mW V4.08	9.20V Idle Tx **9.00V** S6	32mA Idle Tx 110mA	144.225 / **HJ3ZVX5N** Off:13 ID: **715030**	256K 4096K

Figure 16.3: FOX HUNT Check Sheet

Use the **Event Validation Code** column to verfiy that the hunter actually found the transmitter.

Pay attention to the **FRAM/FLASH** column to see that the FRAM and FLASH devices are of adequate size to hold the sequencing commands and the audio files you will use for your hunt. The column is expressed in units directly from the manufacturer datasheet, that is to say in bits. To arrive at command.sequence record count divide bit size by 256 (8 bit bytes, 32 byte records). To estimate audio time, divide the bit size by 36,000 (8 bit samples and 4000 or 5000 samples per second).

In this example, **FOX24** has a small FRAM device. This size device holds only 256 command/sequence records. The other transmitters in the group, being much larger, hold 1024 or more records. The small FRAM device (256 or less records) is flagged in **red**.

The FLASH device sizing is handled in a somewhat similar manner. The minimum size device for convenient operation is one that holds at least 2 minutes of audio (i.e. a 4Mb device). A 2Mb device is flagged in orange and a 1Mb or smaller device is flagged in red.

16.5 fox_label_A_top

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!

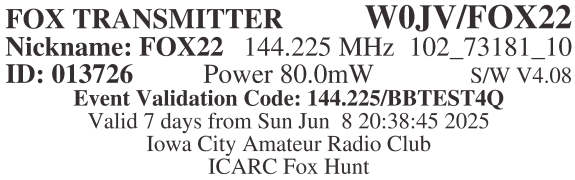


Figure 16.4: FOX Transmitter TOP Labels

These labels are placed on the removeable enclosure cover. These labels are generated before each hunt and affixed to the trasmitter. The labels have ID values that generated when the label utility runs.

The **fox_label_A_top** 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.

You will need to perform some shell-script gymnastics to setup a hunt that requires more than 14 Fox Transmitters. The *fox_label* utility eliminates all existing **fox_label** files before generating new files.

16.6 fox_label_C_FOX*, Serialized Finder Card

These labels are printed on cardstock to be left with the transmitter to be picked up by the hunters. The **fox_label_A_top** and **fox_label_A_chk** files must be generated in the same run to keep the serial numbers synchronized!

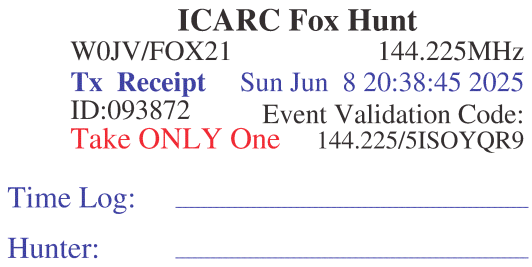


Figure 16.5: FOX Transmitter Found Cards

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_top** file (14 of them).

Simply print a sheet for each transmitter, split them out and rubberband them to the (correct) transmitter. Hunters then take a (single) card from each transmitter to document that they did, in fact, find them.

16.7 fox_label_B_cards; Tx Found Log Card

The *Capture Cards* can be printed when a larger number of participants are expected. This would be necessary if there aren't enough These cards are not dated or serialized so thay aren't time sensitive. They can be carried forward from one hunt to the next. *Transmitter Found Cards* to accommodate all the participants.

These are printed on 4-up post-card stock.

ICARC Fox Hunt
Transmitter Capture LOG (V3.13)

Hunter Callsign

Hunt Date **Card**

Nick	ID	Valid Code
FOX		
FOX		
FOX		
FOX		
FOX		
FOX		
FOX		
FOX		
FOX		

Figure 16.6: FOX HUNT Capture Card

The participant can fill in their callsign, the date of the hunt, and a card index (in the event more than 9 transmitters are used).

Then, for each transmitter located, the transmitter number (as we call the transmitters FOX0 through FOX*nn*), and the validateion codes listed on the transmitter labels.

16.8 fox_label_B_quick_cards; Quick Found Log Card

These *Capture Cards* can be printed for an informal hunt. This eliminates the need to print the *Transmitter Found Cards* on page 287

These are printed on 4-up post-card stock like the *fox_label_B_cards*.

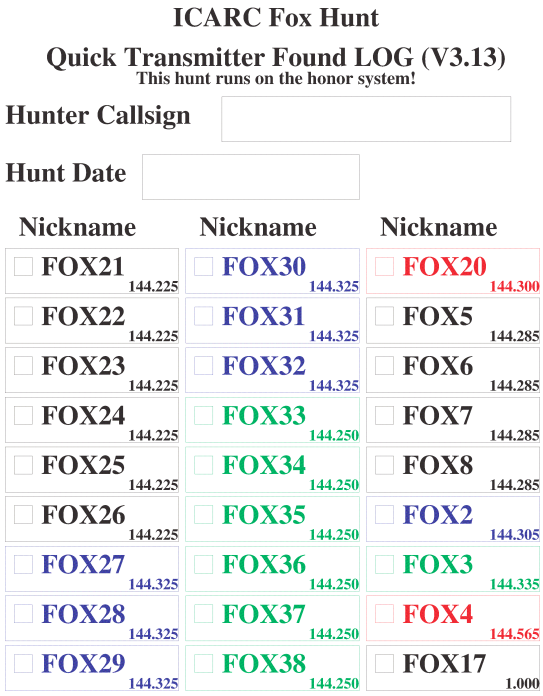


Figure 16.7: FOX HUNT Quick Found Card

The color of the *Nickname* field arranges the fox transmitters into their hunt groups.

The fox *Nicknames* are derived from the *fox_label.csv* file. The operating frequency is also stored in this file.

16.9 fox_label.csv

The transmitter information file.

Listing 16.1: fox_label.csv

```
# LAST LOAD: updated 2025-Jun-08T20:38:35 by /home/wtr/Radio/halo_term/fox_clock
#Fox,   call,    operate, startup,  FRAM,   FLASH, S/W-V,  transmitter/amplifier,  Syn-Xtal, TxPWR, RUN, Id-V, Id-I, Tx-V, Tx-
      ↪ I,Toff, IP-Addr,       MAC-Address
FOX21,  W0JV,    144.225, 144.150,  256K,   4096K, V4.08, 102_73181_10/102_73181_28, 20.000,  70.0, S6,  8.0,   24,  7.6,
      ↪ 122, Off:1
FOX22,  W0JV,    144.225, 144.150,  256K,   8192K, V4.08, 102_73181_10/102_73181_28, 20.000,  80.0, S6,  8.9,   38,  8.6,
      ↪ 123, Off:20
FOX23,  W0JV,    144.225, 144.150,  256K,   8192K, V4.08, 102_73181_10/102_73181_28, 20.000,  95.0, S6,  7.9,   54,  7.2,
      ↪ 157, Off:13
FOX24,  W0JV,    144.225, 144.150,  256K,   4096K, V4.08, 102_73181_10/102_73181_28, 20.000,  60.0, S6,  9.2,   32,  9.0,
      ↪ 110, Off:13
FOX25,  W0JV,    144.225, 144.150,  256K,   4096K, V4.08, 102_73181_10/102_73181_28, 20.000,  50.0, S6,  8.8,   37,  8.6,
      ↪ 123, Off:22
FOX26,  W0JV,    144.225, 144.150,  256K,   8192K, V4.08, 102_73181_10/102_73181_28, 20.000,  80.0, S6,  8.2,   32,  7.8,
      ↪ 123, Off:23
#        1         2         3         4         5         6         7         8         9         0         1         2
#12345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
#
FOX27,  W0JV,    144.325, 144.150,  512K,  65536K, V4.08, 102_73181_10/102_73181_28, 20.000, 115.0, S0,  8.6,   29,  8.3,
      ↪ 121, Off:26
FOX28,  W0JV,    144.325, 144.150,  512K,  65536K, V4.08, 102_73181_10/102_73181_28, 20.000, 100.0, S0,  8.0,   32,  7.5,
      ↪ 118, Off:16
FOX29,  W0JV,    144.325, 144.150,  512K, 262144K, V4.08, 102_73181_10/102_73181_28, 20.000, 100.0, S0,  6.7,   46,  5.4,
      ↪ 171, Off:-86391
FOX30,  W0JV,    144.325, 144.150,  512K, 262144K, V4.08, 102_73181_10/102_73181_28, 20.000, 100.0, S0,  7.8,   47,  7.2,
      ↪ 151, Off:16
FOX31,  W0JV,    144.325, 144.150,  512K,  65536K, V4.08, 102_73181_10/102_73181_28, 20.000,  50.0, S0,  7.6,   39,  7.1,
      ↪ 136, Off:1
FOX32,  W0JV,    144.325, 144.150,  512K,  65536K, V4.08, 102_73181_10/102_73181_28, 20.000,  85.0, S0,  7.4,   40,  6.2,
      ↪ 159, Off:41
#
FOX33,  W0JV,    144.250, 144.150,  256K,  65536K, V4.08, 102_73181_10/102_73161_28, 20.000,  35.0, S8,  8.7,   27,  8.5,
      ↪ 104, Off:26
FOX34,  W0JV,    144.250, 144.150,  512K,  65536K, V4.08, 102_73181_10/102_73161_28, 20.000,  60.0, S9,  7.7,   18,  7.3,
      ↪ 120, Off:0
FOX35,  W0JV,    144.250, 144.150,  512K,  65536K, V4.04, 102_73181_10/102_73161_28, 20.000,   5.0,   ,  0.0,    0,  0.0,
      ↪ 0, ?
FOX36,  W0JV,    144.250, 144.150,  256K,  65536K, V4.08, 102_73181_10/102_73161_28, 20.000,  60.0, S0,  9.6,   38,  9.4,
      ↪ 115, Off:21
FOX37,  W0JV,    144.250, 144.150,  512K,  65536K, V4.08, 102_73181_10/102_73161_28, 20.000, 120.0, S0,  9.9,   34,  9.7,
      ↪ 102, Off:-9
FOX38,  W0JV,    144.250, 144.150,  512K,  65536K, V4.08, 102_73181_10/102_73161_28, 20.000,  60.0, S0,  8.8,   26,  8.6,
      ↪ 106, Off:24
#
```

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 over the first 8 records. In the example **fox_label.csv** file, FOX29 would be the starting point with **-o 8**.

16.9.1 fox_label.csv; First Line

The first line of the csv file **must** be formatted as shown. The *fox_clock* utility updates data in this file as the TOY clock is set prior to the hunt.

The 1st. 12 12 characters will cause the *fox_clock* utility to update the date on this line to reflect when it accessed the file.

16.9.2 fox_label.csv; Column 1

FOX Unit Number or name.

This number is somewhat arbitrary. Units are just numbered in sequential order. Breaks in the sequence are allowed.

16.9.3 fox_label.csv; Column 2

Station Callsign.

This is the callsign that appears on the labels.
When a callsign (using the **-C** flag) is specified this field is ignored.

This should match with the **INI=CALL** *<callsign>* record in the fox transmitter file system.

16.9.4 fox_label.csv; Column 3

Operating Frequency.

This is the frequency the fox transmitter will use during the hunt.

16.9.5 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 on this frequency.

16.9.6 fox_label.csv; Column 5

FOX Unit FRAM Size.

This is the size of the FRAM device as reported in the startup banner.

16.9.7 fox_label.csv; Column 6

FOX Unit FLASH Size.

This is the size of the FLASH device as reported in the startup banner.

Older hardware shows **NULL** to indicate that there is no FLASH device present.

16.9.8 fox_label.csv; Column 7

zNEO software revision.

The software revision of the operating software that operates the Fox Transmitter.

16.9.9 fox_label.csv; Column 8

Transmitter hardware revision and daughterboard hardware revision.

These are the drawing numbers for the main transmitter board and the attached RF daughterboard. The two drawing numbers may be seperated using a slash (/) character.

16.9.10 fox_label.csv; Column 9

Reference crystal on the RF synthesizer (expressed in MHz).

This is the crystal used by the ICS525/ICS307/SI5351.

16.9.11 fox_label.csv; Column 10

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.

16.9.12 fox_label.csv; Column 11

Active RUN Schedule

The clock setting utility captures the schedule that is activated in the **ANN=** file and records it here.

The active schedule is then printed on the check sheet to confirm that the fox skulk is correctly setup.

16.9.13 fox_label.csv; Column 12

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.

These columns in the *.csv* file can be updated as the TOY Clock is updated just prior to the hunt.
Labels can be printed after the *.csv* file has been updated.

16.9.14 fox_label.csv; Column 13

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.

16.9.15 fox_label.csv; Column 14

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.

16.9.16 fox_label.csv; Column 15

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.

16.9.17 fox_label.csv; Column 16

TOY clock offset.

The *fox_clock* utility captures the time value from the second **TIME** command and calculates an offset from the host time. This is all done on a seconds-of-day basis, to accommodate a target Fox Transmitter that is using a truncated UNIX time.

16.9.18 fox_label.csv; Column 17&18

Raspberry-PI Zero-W Network Settings

These columns will show the assigned IP (assuming the host network assigns a static IP) and the Raspberry-PI MAC address.

These appear only for the Raspberry-PI based hardware.

Chapter 17

Synthesizer configuration utilities

These are the utilities used to build the configuration tables for the RF synthesizer chips.

17.1 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.
Although these commands may be placed into the INI= file or the S*n*= file for field use, a better scheme is described in section 5.6 on page 104.

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. In practice, we use an entry in the internal 5351 setup table to measure the carrier offset (from nominal) and then use this error to the build an externally loaded table. See section 5.6 on page 104 for details on managing an external frequency table for the SI5351.

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.

17.1.1 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.

We also have fab'd a filter test jig (102-73181-60) that can be used in place of the RF daughterboard to route the output of the SI5351 to an SMA connector.

17.1.2 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 17.1: 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 17.2: 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 17.3: si5351a_calc-180

```
//                                                                      180
//  Start by calculating the output to input frequency ratio.           181
//    We need this to come up with the VCO frequency                    182
//                                                                      183
abcdef[VCO_MULT]    = fout / xtal;                                      184
```

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 17.4: si5351a_calc-185

```
//                                                                      185
//  Get the VCO to run between 600MHz and 900MHz                        186
//                                ------     ------                     187
//    So get the product of the multiplier from above                   188
//    and an integer division of the VCO frequency                      189
//    (and a power-of-2, if possible)                                   190
//                                                                      191
abcdef[FB_DIV] = 0;                                                     192
abcdef[VCO_FREQ] = 0.0;                                                 193
//                                                                      194
// Loop through even "abcdef[FB_DIV]" ratios                            195
//                                                                      196
while(abcdef[VCO_FREQ]<=900.0E6){                                       197
    abcdef[VCO_FREQ] = xtal * abcdef[VCO_MULT] * abcdef[FB_DIV];        198
    if( (abcdef[VCO_FREQ]>=600.0E6) )                                   199
        break;                                                          200
    abcdef[FB_DIV]+=2;                                                  201
}                                                                       202
```

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 17.5: si5351a_calc-204

```
//  If we didn't find a "pretty spot", where the                    204
//    VCO will be happy, try again using odd divisors               205
//                                                                  206
if(abcdef[VCO_FREQ]>900.0E6){                                       207
    abcdef[FB_DIV] = 1;                                             208
    abcdef[VCO_FREQ] = 0.0;                                         209
    while(abcdef[VCO_FREQ]<=900.0E6){                               210
        abcdef[VCO_FREQ] = xtal * abcdef[VCO_MULT] * abcdef[FB_DIV];  211
        if( (abcdef[VCO_FREQ]>=600.0E6) )                           212
            break;                                                  213
        abcdef[FB_DIV]+=2;                                          214
    }                                                               215
}                                                                   216
```

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 17.6: si5351a_calc-217

```
//                                                                  217
//  If the abcdef[VCO_FREQ] exceeds 900MHz, we've failed, and miserably  218
    ↪ at that
//  we'll notice it later and zero out the control words...         219
//                                                                  220
if(flag){                                                           221
    fprintf(stdout, "\n");                                          222
    fprintf(stdout, "        ");                                    223
    fprintf(stdout, "xtal %.3f  ", xtal/1.0E6);                     224
    fprintf(stdout, "trgt %.3f  ", fout/1.0E6);                     225
    fprintf(stdout, "divs %.3f  ", abcdef[FB_DIV]);                 226
    fprintf(stdout, "mult %.3f  ", abcdef[VCO_MULT]);               227
    fprintf(stdout, "VCOf %.3f  ", abcdef[VCO_FREQ]/1.0E6);         228
    fprintf(stdout, "\n");                                          229
}                                                                   230
```

Diagnostics. This emits a report if the flag variable is non-zero.

Listing 17.7: si5351a_calc-231

```
//                                                                    231
//  Here we're going to try to come up with the                       232
//  "Synthesis, Stage 1" register values                              233
//                                                                    234
//  the "a" field is the integer portion of the                       235
//    xtal multiplier.  b/c is the fraction.                          236
//                                                                    237
//    fa = modf(abcdef[VCO_MULT], &abcdef[AIDX]);                     238
//                                                                    239
//   abcdef[AIDX].fa is the xtal multiplication factor                240
//                                                                    241
fa = modf(abcdef[VCO_MULT]*abcdef[FB_DIV], &abcdef[AIDX]);          242
```

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 17.8: si5351a_calc-243

```
//                                                                    243
//  fa is the fractional part, so calculate                           244
//  b/c = fa                                                          245
//                                                                    246
switch(2){                                                          247
    case 1:                                                         248
        abcdef[BIDX] = fa * K1048575;                               249
        abcdef[CIDX] = K1048575;                                    250
        break;                                                      251
    case 2:                                                         252
        abcdef[BIDX] = fa * Kvalue;                                 253
        abcdef[CIDX] = Kvalue;                                      254
        break;                                                      255
}                                                                   256
```

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 17.9: si5351a_calc-257

```
//                                                                    257
//  OK, now we have the "a", "b", and "c" numbers to                  258
//   calculate the register values.  AN619-3 S-3.2                    259
//                                                                    260
abcdef[BCFRAC] = abcdef[BIDX]/abcdef[CIDX];                         261
abc_fvco = xtal * (abcdef[AIDX] + abcdef[BCFRAC]);                  262
```

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 17.10: si5351a_calc-263

```
//                                                                    263
//  We will use this "floor" function twice, so do it once           264
//    here so it doesn't get mangled accidentally later...            265
//                                                                    266
bc128_floor = floor(128.0 * abcdef[BCFRAC]);                        267
```

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 17.11: si5351a_calc-268

```
//                                                                    268
//  Calculation for the 1st. Multisynth: AN619-3 S-3.2                269
//                                                                    270
MSNx_P[1] = 128.0 * abcdef[AIDX] + bc128_floor - 512;               271
MSNx_P[2] = 128.0 * abcdef[BIDX] - (abcdef[CIDX] * bc128_floor);    272
MSNx_P[3] = abcdef[CIDX];                                           273
```

0

Now we can build the values that will be loaded into the MSNA and MSNB registers.

Listing 17.12: si5351a_calc-274

```
//                                                                    274
//  OK< the 1st. Multisynth values are saved...                       275
//                                                                    276
if(flag){                                                           277
    fprintf(stdout, "   MUL  ");                                    278
    fprintf(stdout, "abcdef[AIDX](%.3f) abcdef[BIDX](%.3f) abcdef[CIDX  279
        ↪ ](%.3f)  ", abcdef[AIDX]+fa, abcdef[BIDX], abcdef[CIDX]);
    fprintf(stdout, "floor(%.3f)  ", bc128_floor);                  280
    fprintf(stdout, "fvco(%.3f)  ", abc_fvco/1.0E6);                281
    fprintf(stdout, "\n");                                          282
    fprintf(stdout, "        ");                                    283
    fprintf(stdout, "MSNx_P[1] %.3f  ", MSNx_P[1]);                 284
    fprintf(stdout, "MSNx_P[2] %.3f  ", MSNx_P[2]);                 285
    fprintf(stdout, "MSNx_P[3] %.3f  ", MSNx_P[3]);                 286
    fprintf(stdout, "\n");                                          287
}                                                                   288
```

Listing 17.13: si5351a_calc-289

```
//                                                                    289
//  2nd. Multisynth calculations.                                     290
//  Another instantiation of this block,                              291
//    so the calculations are the same, just                          292
//    using a different divisor.                                      293
//  RENAME a, b, c to d, e, f so we don't                             294
//    overwrite previous work...                                      295
//                                                                    296
//  We've built upon the assumption that this                         297
//    Multisynth is operating in integer mode,                        298
//    so the d and e numbers specify zero.                            299
//                                                                    300
abcdef[DIDX] = abcdef[FB_DIV];                                      301
abcdef[EIDX] = 0;                                                   302
abcdef[FIDX] = K1048575;                                            303
abcdef[EFFRAC] = abcdef[EIDX] / abcdef[FIDX];                       304
```

The MS0, MS1, and MS2 *Multisynth*s are setup the same. The values they are loaded with are calculate din the same manner as the MSNA and MSNB *Multisynth*s 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 17.14: si5351a_calc-305

```
//                                                                    305
//  That floor function, from above...                                306
//                                                                    307
def_floor = floor(128. * abcdef[EFFRAC]);                           308
```

As we did above, the FLOOR calculation is done once here and used in several locations.

Listing 17.15: si5351a_calc-309

```
//                                                                      309
//  Calculation for the 2nd. Multisynth: AN619-5 S-4.1.2                310
//    We're operating in the 2M band, so we WILL stay below 150MHz      311
//      so make use of the same equation set using different data.      312
//  Since this is an integer division, the [2] field should calculate   313
//  as ZERO.  Later we will change the [3] field to 1 so it looks nice. 314
//                                                                      315
MSx_P[1] = 128.0 * abcdef[DIDX] + def_floor - 512;                      316
MSx_P[2] = 128.0 * abcdef[EIDX] - abcdef[FIDX] * def_floor;             317
MSx_P[3] = abcdef[FIDX];                                                318
```

Now the numbers for the three *Multisynth* registers can be calculated. This is the same as the MSNA/MSNB *Multisynth*s.

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 17.16: si5351a_calc-319

```
//                                                                      319
//                                                                      320
//                                                                      321
if(flag){                                                               322
    fprintf(stdout, "   DIV  ");                                        323
    fprintf(stdout, "d(%.3f) e(%.3f) f(%.3f)  ", abcdef[DIDX], abcdef[  324
        ↪ EIDX], abcdef[FIDX]);
    fprintf(stdout, "floor(%.3f)  ", def_floor);                        325
    fprintf(stdout, "\n");                                              326
    fprintf(stdout, "        ");                                        327
    fprintf(stdout, "MSx_P[1] %.3f  ", MSx_P[1]);                       328
    fprintf(stdout, "MSx_P[2] %.3f  ", MSx_P[2]);                       329
    fprintf(stdout, "MSx_P[3] %.3f  ", MSx_P[3]);                       330
    fprintf(stdout, "\n");                                              331
}                                                                       332
```

Diagnostics.

Listing 17.17: si5351a_calc-333

```
//                                                                      333
//  Last minute sanity checks.                                          334
//    If we try to drive the internal VCO past 900MHz                   335
//      we are operating out-of-spec, so indicate a                     336
//      problem by zero-ing out all the counters                        337
//                                                                      338
//    If we are "in-spec" copy the results to                           339
//      the callers buyffers.                                           340
//                                                                      341
if(    (abcdef[VCO_FREQ]<=900.0E6) &&                                   342
       (abcdef[VCO_FREQ]>=600.0E6) ){                                   343
    //                                                                  344
    //  1st. Multisynth                                                 345
    //                                                                  346
    ms_values[RFZERO_MSNx_P1] = MSNx_P[1];                              347
    ms_values[RFZERO_MSNx_P2] = MSNx_P[2];                              348
    ms_values[RFZERO_MSNx_P3] = MSNx_P[3];                              349
    //                                                                  350
    //  2nd. Multisynth                                                 351
    //                                                                  352
    ms_values[RFZERO_MSx_P1] = MSx_P[1];                                353
    ms_values[RFZERO_MSx_P2] = MSx_P[2];                                354
    ms_values[RFZERO_MSx_P3] = MSx_P[3];                                355
```

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 17.18: si5351a_calc-356

```
    //                                                                  356
    //  2nd. Multisynth fraction control:                               357
    //    we're integer divisionj, so fraction is                       358
    //    set to ZERO by making [2]=0 and [3]=1                         359
    //    i.e. 0/1 for a fractional part of the divisor                 360
    //                                                                  361
    if(!ms_values[RFZERO_MSx_P2])                                       362
        ms_values[RFZERO_MSx_P3] = 1;                                   363
```

Now for a bit of cleanup. The MS0..MS0 *Multisynth*s 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 17.19: si5351a_calc-364

```
        //                                                                  364
        //  The R Dividers (AN619-6 S-4.2.2)                                365
        //    are set to "divide by 1" using an                             366
        //    index of 0.                                                   367
        //  This is because we are operating                                368
        //    well above the 500KHz point where these                       369
        //    dividers become necessary.                                    370
        //                                                                  371
        ms_values[RFZERO_RX_DIV]   = 0;                                     372
```

Returning more diagnostic information.

Listing 17.20: si5351a_calc-373

```
        //                                                                  373
        //  The MSx_DIVBY4[1:0] is set to ZERO                              374
        //    because we are operating below 150MHz                         375
        //    (144MHz to 148MHz)                                            376
        //                                                                  377
        ms_values[RFZERO_MSX_DIVBY4]   = 0;                                 378
```

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 17.21: si5351a_calc-379

```
        //                                                                  379
        //  An finally, pass the integer feedback                           380
        //    divisor we discovered at the begining.                        381
        //                                                                  382
        sts = abcdef[FB_DIV];                                               383
        //                                                                  384
```

Diagnostics.

Listing 17.22: 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 17.23: si5351a_calc-396

```
    return sts;                                                    396
}  //  kc0jfq_setup()                                             397
```

Return the divisor we planned to stick into the MS0..MS0 *Multisynth*.

17.2 ICS307

Insufficient stock on hand to fabricate the 102-73181-0 boards. No additional documentation at this time.

par

17.3 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.

The Renasas parts (ICS307 and ICS525) are very limited in the frequencies in the 2M band they can generate. A large number of generated frequencies fall between the 5KHz channels that a typical HT can deal with.

Chapter 18

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...

18.1 102-73161-12 Transmitter Configuration

The transmitter may be built in of several configurations. The 102-73161-12 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.

18.1.1 102-73161-12 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.

18.1.2 102-73161-12 MAX2602

R31 is still unpopulated , C62, C67 and R12 are installed.

This is the configuration as originally designed.

18.1.3 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.

18.1.4 Transmitter Configuration, MMIC

Applies to 102-73161-21 and 102-73161-23 daughter boards for the 102-73161-25 and 102-73181 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.

18.1.5 102-73161-12 Amplifier Patch Board, NPN in SOT23 package

This is a (failed) attempt to patch in an simple NPN transistor in place of the MAX2602.

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.

18.2 102-73161-12 Frequency Selection

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.

18.3 102-73161-25 Frequency Selection

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.

18.4 Garbled Audio

If you encounter what sounds like noise or garbled audio consider that the FLASH loading process does not check for existing data before starting a programming cycle.

If you reload the FLASH memory or append new audio to existing audio and overwrite data several issues can occur.

Also keep in mind that you will typically manage the **TALK=** directory seperately from the audio waveform data. A mis-match here can cause problems.

18.4.1 Missing Load (Empty FLASH)

If audio data is missing, all you get is silence as the nominal file image has RIFF/WAVE headers the describe the sample rate, sample width, and sample count. With no header, and no additional information in the **TALK=** directory entry, all that occurs is the handler returns immediately and the sequencer continues on with the next command in the sequence.

18.4.2 Overwritten RIFF/WAVE Headers

If you forget to erase the FLASH prior to loading, you may hear silence when the RIFF/WAVE header data is damaged. If the header key ("RIFF" in the first 4 bytes pointed to in the **TALK=** directory, you will get silence. If the sample rate or sample count fields are corrupt you may hear garbled audio or no audio at all.

18.4.3 Overwritten Waveform Data

Another side-effect of failing to erase the FLASH prior to loading, where only the data area has been overwritten, is garbled audio. This is data dependant and you may hear audio under a high noise floor. You may hear random noise.

18.4.4 Mismatched TALK= Directory

Since the **TALK=** directory is stored in the file that is loaded into the FRAM, you may get the **TALK=** directory out of sync with the waveform data.

18.5 Corrupt Sequences

You may find that some sequences seem to have been corrupted. This may be a side-effect of having a small FRAM device installed.

If you are loading a comprehensive set of sequences, such as multiple operating sequences, into an FRAM that is too small, some of the sequence will be lost. Needless to say, if there is not adequate room in the FRAM for the size of the load it won't work too well.

The transmitter log seen in section 16.9 on page 290 shows *FOX24* has been fabricated with a relatively small FRAM. This 64KB device holds 256 records and the image loaded into FRAM is 286 records (about 73Kb), too large for the 64Kb device.

To allow this unit to fully operate as a member of the hunt group that consists of FOX21..FOX26, the external frequency table has been limited to 25KHz steps (rather than 5KHz steps). This doesn't affect group operations as the 25KHz channel step covers target frequencies while leaving the sequences available to the group untouched. The records we may want to alter are also left in the same position as the other units as the external frequency table is loaded at the end of the FRAM.

18.6 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.

18.7 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.

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.

The original plan was to be able to synchronize time in the field. This time networking feature has been deprecated as it was never practical in the field. Furthermore the serial port used to configure the transmitter was moved to the 3.5mm jack that is externally accessible.

18.8 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 with abnormal intercharacter 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.

18.9 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 187).

18.10 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.

18.11 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).

18.12 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.

18.13 Battery Check

Assuming you are using an announce message similar to the example in section 19.1.6 on page 324, you should hear a battery report as part of the sign-on message. Keep in mind that all transmitters should normally be be configured to use a single sign-on reporting frequency. Cycling power will cause the system to start by running the **INI=** commands followed by the **ANN=** commands.

The **ANN=** commands should include a battery report, either in code, encoded, or as a plain voice report (depending on your specific configuration). If in doubt about the status of a unit, a simple cycling of the power switch should let you know if the unit is working.

The internal TOY clock will keep the unit synchronized with all other units. There should be no need to worry about time synchronization if the TOY clock is working.

18.14 Alternate Battery Configurations

In general, the author has foud that using disposable batteries provides the most convenient mode of powering the Fox Transmitters. In particular, when setting up a large *skulk* (skulk is the name for a group of foxes), replacing batteries is far less time consuming than trying to manage rechargeable cells. As of April 2025, I haven't found a lithium chemistry battery pack that will fit in the case that wouldn't require charging before each and every event. With 18 or more transmitters, it is time consuming enough to simply connect each unit to the host computer to update the TOY clock.

If you would like to make use of rechargeable cells, there are some features present on the all revisions of the board to allow connecting a charge port.

Near the ON/OFF switch (SW2) is an unpopulated connector, J7, that gives you access to the batterys when the unit is switched off. This is the same connector series as the one in the J2 position used to connect the battery pack. The mating connector locks in to both the right-angle (J2) and the vertical (J7) connectors.

When the unit is switched off J7 and J2 are connected to provide a charge path.

For an external charging jack, there are a few 2.5x5.5 bulkhead mount connectors found on DigiKey.

Vendor	Number	Hole	Mount
Same Sky	PJ-011B	13mm mounting hole	rear mount
Same Sky	PJ-065B	5/16" mounting hole	rear mount
GlobTek	JACK-L-PANEL-8A(R)	11.2mm mounting hole	front mount
Tensility Int.	54-00064	11mm mounting hole	front mount
Switchcraft	712A	11mm mounting hole	rear mount
Switchcraft	L712A	5/16" mounting hole	rear mount

Table 18.1: CONN PWR JACK 2.5X5.5MM SOLDER

Parts with the *rear mount* designation can be assembled on the bench and then installed into the enclosure.

Parts with the *front mount* designation must be assembled in the enclosure. Make sure you have any washers and the nut slipped on to the cable before assembly.

Also note that the J7 pinouts match the J2 pinouts. These connectors use identical mating connectors and swapping them around will not cause a problem, other than switching the ON and OFF/CHARGE positions for the switch.

18.14.1 Higher Voltage Packs

You may choose to employ higher voltage packs. A 3-cell lithium pack, operating at about 11 volts, can be employed by changing the voltage monitor resistor R35. Consult the schematic and review the coefficients tables (**BATR I**) to select an appropriate value for R35.

Using higher voltage packs will nominally be OK as long as you are using only regulated 5 volts on the RF daughterboard, and you don't exceed the maximum input voltage rating of the first stage regulator. Keeping the nominal pack voltage below 24V should work well with all of the available switchmode regulators.

Some of the 5 volt regulator parts will accomodate up to 36 volts.

The SI3865 power switch in position U91 only allows for the use of up to a 12V battery. U81 switches the regulated 5 volt line so is never a problem. U91, however, should not be populated if you plan to exceed the parts 12 volt rating.

If you can find it, an SI3861 will accommodate up to 20 volts and is pin compatible. This is an obsolete part and will not generally be available.

If you are using an RF amplifier that runs directly off of the battery, and plan to use a higher voltage pack (for higher RF output levels) power switching needs to move to the the RF amplifier daughterboard and the switch at U91 would need to be bypassed.

18.15 Universal Setup Pitfalls

The examples in section 19.1 on page 317 provide commanding for both the SI5351 RF clock and the DRA818/SA818 tranceiver modules. When using the SI5351, the DRA818 configuration selection must be erased. The latest RF generator selection encountered controls the execution path within the zNEO.

An incorrect selection (i.e. you forget to remove the DRA818 configuration entry after loading FRAM) will result in a rather spectacularly non-functional fox transmitter. Check that you updated the **INI=** section in the FRAM. When using the SI5351 in particular, "**EDMP** 818" and then "**ERAS** nn" the DRA818 configuration record .

Do **not** make the mistake of using **EZER nn** to zero the record as this will render everything after that record inaccessible until the next **ESAV** command rewrites that record. The "**ERAS** nn" rewrites the record with a **MT**** command.

18.16 Nulla malesuada

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.

Chapter 19

Actual FOX configuration commands

This is a listing of the setup commands for the prototype units.
Lines without the *esav* are not placed in FRAM.

Updates in the fox operating software and the support utilities provide for command argument substitution. This should feel familiar to a bash shell user.

Parameter substitution is used extensively to allow one set of configuration files to be used to load multiple hunt groups with multiple units in each group.

First order substitutions change the nickname, operating frequency, and operating schedule for each unit. Some additional substitutions alter the files that are included by the main file to further accommodate the diverse set of transmitters we deal with here.

19.1 FOX2X_KC0JFQ setup scripts

This is the current master setup for all of the ICARC FOX transmitters. The **fox_simple** utility has some features/additions to make sharing these *fox scripts* a bit easier.

Every effort has been made to keep the commanding identical to the previous software version to avoid obsoleting old scripts.

With the introduction of the V4.0 software (that provides a high speed binary loading protocol), we are moving all setup commands into the **FOX2X_KC0JFQ.fox** setup. This allows a complete log of the commands that are to be downloaded into the target to appear in the log file

The log file can then be used to perform a fast load of the FRAM. This improves load speed dramatically.

19.1.1 FOX2X_KC0JFQ.fox

Listing 19.1: FOX2X_KC0JFQ

```
REM- 678901234567890123456789 1
esav REM- ./'fox_simple' 2
esav REM- -S 'usb' 3
esav REM- -F 'filename' 4
esav REM- 'fdate' 5
# 6
# 7
# /home/wtr/Radio/halo_term/fox_simple 8
# -fFOX2X_KC0JFQ.fox -lFOX2X_KC0JFQ.log 9
# -Xchrp1=6,0 -Xchrpfrq=1.0 10
# -Xsched=S0 -Xtone=1.0 11
# -Xstooge=120 -Xchirp_up=0 12
# -Xchirp_dn=0 -Xruns6=600,300 13
# -Xsynth_dev=SI5351 -Xsynth_set1=8MA 14
# -Xsynth_set2=CLK0 -Xtalk_file=talk_73181_rxxk.fox 15
# -Xsx_stooge=SX_STOOGE.fox -Xspare1=not 16
# -Xspare2=used -Xbatvc=BATV 17
# -CW0JV -NFOX24 18
# -R360,180 -Q144.285 -Afreq_5351-08.fox 19
# 20
# /home/wtr/Radio/halo_term/fox_simple -SFOX2X -c-50 -t10 -f/home/wtr/WAV/ 21
    ↪ fox_73181_rxxk.hex
# /home/wtr/Radio/halo_term/fox_simple -b115200 -SFOX2X -c50 -t10 -f/home/wtr/WAV 22
    ↪ /talk_73161_rxxk.hex
# 23
# /home/wtr/Radio/halo_term/fox_binary -F -S FOX2X -a /home/wtr/WAV/ 24
    ↪ talk_73181_2025_TREK.hex
# /home/wtr/Radio/halo_term/fox_binary -F -S FOX2X -f FOX24_KC0JFQ.log 25
# 26
# 27
```

Header lines, that document what we're up to in this fox script.

All these comments are used as sample command lines for all the fox transmitters used by the Iowa City club. A simple cut and paste from here is used to load the individual transmitters.

Comments on lines 2 through 5 are inserted into the FRAM to provide documentation of the file used to load the station and the last modification date on the master file.

All four of these comments use substitutions to send useful information to the fox system.

Line 2 substitutes in the program name for the *'fox_simple'* string.

Line 3 subs in the serial device nickname used to load the fox system for the *'usb'* string.

Line 4 subs in the filename (i.e. FOX2X_KC0JFQ.fox) for the *'filename'* string.

Line 5 subs in the file modification date for the *'fdate'* string.

We can also insert the current date using a *'date'* substitution string.

Listing 19.2: Load External Frequency Table

```
#                                                                    132
#  Move the frequency setup to end of setup so we                    133
#  don't have to search so long for other stuff                      134
#                                                                    135
#include 'off_file'                                                  136
#include 'fm_file'                                                   137
#include 'iaru_80m'                                                  138
```

The **'off_file'** substitution requires the offset table to be named in the controlling shell script. This load occurs after the TALK an operating commands are loaded (observe the line numbers from the source file).

With the V3.72 update, the frequency table is external. This increases the size of the table so we move it down towards the end of the FRAM file system. We move it toward the end to minimize command search time when executing the fox message sequence.

A sample of the included table is found in section 19.3 on page 331.

The 4.07 release adds the capability to operate in the 88-108MHz FM band. Line 137 inserts a table to configure the SI5351 for operations in that band.

Listing 19.3: Enable TOY Clock

```
#                                                                    139
#  Make sure clock is correctly configured!!!                        140
#                                                                    141
TOYC NONE                                                          142
```

Here we disable the charging features of the DS1672. They are not needed as we have an external battery maintenance circuit.

The audio file system, line 33, is nominally identical on all units, although one or two have s small FLASH device, somore than one variant is needed. Audio clips are present in the audio file system for all unit nicknames to allow for the use of a common audio file.

The include directives (lines 33 and 230) tells *fox_simple* to insert the talk directory and supplemental frequency table at those points. This is a convenience for us in managing files. A command file can be inserted anywhere it is needed.

Also note the tickling of the TOY clock at line 43 and 45.

This serves to force the DS1672 into a normal operating mode. This command must be issued after the DS1672 has first been powered up. This enables the oscillator so that it will continue to track time when power is removed.

Sending the command will not cause the DS1672 to stop operating, so it may be issued without hazard here.

19.1.2 FOX2X_KC0JFQ TALK Directory Include

Listing 19.4: FOX2X_KC0JFQ_29

```
#                                                                    29
#  Limited voice storage                                             30
#                                                                    31
# //#include talk_73181_1.fox                                       32
#include 'talk_file'                                                 33
#                                                                    34
```

The controlling shell script is expanding to load most units in the ICARC *fleet*. To that end the older units that lack a FLASH memory to hold audio waveform data need this *'talk_file'* substitution to allow loading an very abbreviated **TALK** directory.

The TALK Directory file may is found in section 19.2 on page 327.

19.1.3 FOX2X_KC0JFQ INI=

This **INI=** sequence is run when zero or one of the TEST/MAS jumpers are installed.

Listing 19.5: FOX2X_KC0JFQ-INI

```
#                                                                    35
#  Tickle the DS1672, if it doesn't respond                          36
#    we'll hit it again later.                                       37
#  If itdoes respond, the 2nd. time we read                          38
#    it will give matching time, so no foul...                       39
#       Our Epoch is CDT: -5 Hours from Zulu                         40
#       Set system time from DS1672                                  41
#                                                                    42
esav INI=TIME                                                        43
esav INI=WAIT 0.5                                                    44
esav INI=TIME                                                        45
esav INI=EPOC -5.0                                                   46
esav INI=NAME 'name'                                                 47
esav INI=CALL 'call'                                                 48
#                                                                    49
#esav INI=CONF BMON 12.5V                                            50
esav INI=CONF 'synth_dev'                                            51
esav INI=CONF 'synth_set1' 'synth_set2'                              52
#esav INI=CONF DRA818                                                53
esav INI=CONF 'spare1' 'spare2'                                      54
esav INI=FREQ 144.150                                                55
esav INI=BATR                                                        56
#                                                                    57
#       Set schedules,  leaving ONLY                                 58
#                                                                    59
REM= 01234567890123456789012345678901234567890                       60
esav INI=MODS S0 'run'                                               61
esav INI=MODS S1 'run'                                               62
esav INI=REM= MODS S2 'run'                                          63
esav INI=REM= MODS S3 'run'                                          64
esav INI=REM= MODS S4 'run'                                          65
esav INI=REM= MODS S5 'run'                                          66
esav INI=MODS S6 'runs6'                                             67
esav INI=MODS S7 'run'                                               68
esav INI=MODS S8 360,0                                               69
esav INI=MODS S9 360,15                                              70
esav INI=STAT                                                        71
#                                                                    72
```

Using the *fox_simple* utility, we substitute for the *'name'*, *'call'*, and *'run'* keys. This moves the unit-specific tailoring to the *fox_simple* utility so the **FOX2X_KC0JFQ.fox** file is the same for all of the 102-73181-xx fox transmitters.

The **TIME** command, lacking any arguments, simply copies the hardware clock into the system clock. It is issued twice to overcome an issue with the DS1672 operating correctly when first queried. The **TIME**, **WAIT**, **TIME** gets the DS1672 working and loads the time into the system reliably.

Our transmitter is now running on truncated UT (within a second or two of all the other fox transmitters).

The **EPOC** command sets the local timezone offset for Central Daylight time (5 hours before ZULU).

Assuming you are creating schedules that evenly divide into sixty minutes (i.e. no remainder), the **EPOC** offset does not affect operation.

Simply put, all transmitters must run with the same **EPOC** offset.

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. The **CONF 'synth_dev'** will be substituted with **CONF SI5351** for the 102-73181-10 boards. We also configure the SI5351 output by changing the The **CONF 'synth_set1' 'synth_set2'** to be **CONF 8MA CLK0**. The **CONF 'spare1' 'spare2'** line may be overwritten (after loading) with **CONF DRA818** to easily switch to that RF subsystem. The **CONF SI5351** and **CONF 8MA CLK0** become superfluous for the DRA818 and are effectively ignored (no harm, no foul).

When we are using the SI5351, the **CONF 8MA CLK0** enables the CLK0 output and sets the drive strength to, in effect, full. This provides a clock to any of the 102-731*1-28 low power amplifier daughter board. The RF clock is delivered directly from the SI5351 on the motherboard to the amplifier board. Selecting **8MA** provides a 50Ω output impedance from the SI5351.

More substitution occur in the **CONF** commands to deal with the older fox transmitters in the *fleet*. With the older transmitters we will substitute in **CONF ICS525**, **CONF VOICE FRAM** and a **CONF BMON 73161** commands to configure for the 102-73161-25 boards. We also have the **BATR** command that reports the idling current after the **CONF BMON** command so we get a correct battery voltage reading.

The next step is to select the transmitter frequency. The selection here is the common setup frequency, **not** the target 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.

19.1.4 FOX2X_KC0JFQ TEST sequence

This TEST may be used as the operator sees fit. As shown, the **TEST** jumper will configure the SI5351 to deliver the crystal oscillator (20MHz) directly to the *CLK1* pin to be used as a clock source for an external project.

Listing 19.6: FOX2X_KC0JFQ_TEST

```
#                                                                    74
#                                                                    75
esav TEST=FREQ 144.100                                               76
esav TEST=BEGN,SILENT                                                77
#esav TEST=5351,XTALT                                                78
#                                                                    79
```

This **TEST=** sequence is run when the *TEST* jumper is in place. The **INI=** sequence is run before the **TEST=** sequence is run.

If everything goes wrong, both the TEST= and MAS= jumpers may be installed to force the system into recovery mode to clear and reload the FRAM as required.

19.1.5 FOX2X_KC0JFQ MAS sequence

This MAS may be used as the operator sees fit. If everything goes wrong, both the TEST= and MAS= jumpers may be installed to force the system into recovery mode to clear and reload the FRAM as required.

Listing 19.7: FOX2X_KC0JFQ_MAS

```
#                                                                    79
#                                                                    80
#include master_2025.fox                                            81
#                                                                    82
#                                                                    83
```

This **MAS=** sequence is run when the *MAS* jumper is in place. The **INI=** sequence is run before the **MAS=** sequence is run.

19.1.6 FOX2X_KC0JFQ ANN=

This **ANN=** sequence is run after the **INI=** sequence when no jumpers (**TEST** or **MAS**) are installed. This is the station aliveness and status message.

Listing 19.8: FOX2X_KC0JFQ_ANN

```
#                                                                    84
#  We're making use of the <CALL> and <NAME> substitution           85
#    inside the fox transmitter!!!                                   86
#                                                                    87
esav REM- fox_ann_V2025.fox                                         88
esav ANN=TONE 1.0                                                   89
esav ANN=CWPM 30,-1,-1,-1,-1                                        90
esav ANN=BEGN                                                       91
esav ANN=BATR                                                       92
esav ANN=TALK <CALL>                                                93
esav ANN=TALK <NAME>                                                94
esav ANN=WAIT 1.0                                                   95
esav ANN='batvc' V 7.2                                              96
esav ANN='batvc' I                                                  97
esav ANN=TIRP ASN                                                   98
esav ANN=WAIT 0.3                                                   99
esav ANN=TALK 'freqM'                                               100
esav ANN=TALK 'freqK'                                               101
esav ANN=TONE 1.0                                                   102
esav ANN=CWPM 30,-1,-1,-1,-1                                        103
esav ANN=DONE                                                       104
esav ANN=FREQ 'freq'                                                105
esav ANN=STAT                                                       106
esav ANN=RUN0 'sched'                                               107
#                                                                   108
```

We make use of the *'freq'* substitutions here to set the operating frequency. We have also moved the selection of the active schedule (i.e. the *'sched'* substitution) to the shell script to avoid the need to edit the fox file or alter it directly in the fox transmitter.

The basic setup for operation on lines 90 and 91. The audio CW tone is set on line 90 and the chipping rate on line 91. 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 **BATR** command serves to provide a battery report when transmitting for logging in the *fox_label.csv* file (see section 16.9 on page 290) before the hunt. The **BATV** commands verbally report the voltage and current for the transmitter, again so the operator can be confident the fox transmitter will remain functional throughout the hunt.

We could use the **BATC** command to report about battery condition using code. **BATC** reports in plain code using numbers or it may encode the voltage or current as a series of **T**s and **E**s; **T**s are volts and **E**s are tenths.

The **TIRP** command is introduced into the V4.06 software release. This command produces a time report to allow the **TOY** clock to be audibly *inspected*. This command synchronizes with the system clock on a 5 second boundary and emits a short tone. The tone is sent at the current audio frequency but the duration is fixed in the **TIRP** command (i.e. unaffected by **CWPM** settings).

CWPM settings). Bring up a clock (with a seconds display) on a GPS or cellphone when powering on the Fox Transmitter. When the TIRP command sends the *beep*, compare it with the reference clock. Use the *beep* to determine if the Fox Transmitter is running fast or slow.

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 may be 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.

19.1.7 FOX2X_KC0JFQ sequence includes

The individual message sequences have been moved to individual file to allow for easier updates. Those sequences are included into this document, so externalizing them reduces changes to the *FOX2X_KC0JFQ.fox* file and keeps the included file line number more static.

Listing 19.9: FOX2X_KC0JFQ 108

```
#                                                                    108
#  ICARC prototypical operating scenario                             109
#                                                                    110
#include FOX2X_S0.fox                                               111
#                                                                    112
#  Formal FOX hunt sequences                                         113
#    replaces S1= through S5=                                        114
#                                                                    115
###include S_MOxx.fox                                               116
###include S_sprint.fox                                             117
#                                                                    118
#  Other fun scenarios                                               119
#                                                                    120
#include FOX2X_S1.fox                                               121
#include FOX2X_S2.fox                                               122
#include FOX2X_S3.fox                                               123
#include FOX2X_S4.fox                                               124
#include FOX2X_S5.fox                                               125
#include FOX2X_S6.fox                                               126
#include FOX2X_S7.fox                                               127
#                                                                    128
#  A bit of levity                                                   129
#                                                                    130
#include 'sx_stooge'                                                131
```

Added in late May of 2025 are IARU compatible operating scenarios. We have both a standard schedule where each transmitter is allocated one minute out of 5 and a sprint schedule where4 eah transmitter is allocates 12 seconds out of eacyh minute.

The standard schedule emits the standard station identification at the begining and end of each message. The word rate for the ID is a bit faster at 20 WPM.

The sprint schedule emits only the station callsign at the begining of the message. The station callsign is sent at 20 WPM. The message rates vary to allow 3 repetitions of the hunt identifier in each on-air segemnt.

The IARU scenario replaces the S1=..S5= sequences and requires editing the **FOX2_KC0JFQ.fox** file to remove **FOX2X_S1.fox** through **FOX2X_S5.fox** and insert either **S_MOxx.fox** or **S_sprint.fox**.

Also note that **CODE <CALL>** and **CODE <NAME>** will not be properly substituted with the actual callsign and nickname prior to version V4.09.

As shown in the example the **S_MOxx.fox** and **S_sprint.fox** includes, with the extra "##", aren't loaded. And and remove the "##" as needed.

19.2 TALK Directory file

This is a list of the audio clips that are loaded into FLASH memory. This assumes that software version **V3.27** or 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 19.10: talk_73181_2025_TREK.fox

```
esav ID=FL talk_73181_2025_TREK.fox
esav ID=FL 2025-06-03T14:20:27
esav TALK=KC0JFQ 0
esav TALK=W0JV 5760
esav TALK=FOX20 11904
esav TALK=FOX21 14848
esav TALK=FOX22 19072
esav TALK=FOX23 23424
esav TALK=FOX24 28672
esav TALK=FOX25 33408
esav TALK=FOX26 39552
esav TALK=FOX27 44544
esav TALK=FOX28 49152
esav TALK=FOX29 53632
esav TALK=FOX30 58624
esav TALK=FOX31 62464
esav TALK=FOX32 67200
esav TALK=FOX33 72064
esav TALK=FOX34 77696
esav TALK=FOX35 83328
esav TALK=FOX36 88832
esav TALK=FOX37 94208
esav TALK=FOX38 99328
esav TALK=FOX39 103552
```

These are the callsign and nickname voice clips.

The filenames **must** match what appears on the **TALK** command; this is a simple string match operation!

Note that verbalizations for all units *nickname*s is stored in every unit. This makes audio file management simpler and there is plenty of FLASH available.

Listing 19.11: talk_73181_2025_TREK.fox_25

```
esav TALK=BATTI 108928
esav TALK=BATTV 113152
esav TALK=REG5 117632
esav TALK=POINT 122752
esav TALK=V_HZ 124160
esav TALK=V_KHZ 126592
esav TALK=V_MHZ 129792
esav TALK=V_N0 132992
esav TALK=V_N1 135680
esav TALK=V_N2 137472
esav TALK=V_N3 139648
esav TALK=V_N4 141568
esav TALK=V_N5 143488
esav TALK=V_N6 145664
esav TALK=V_N7 147456
esav TALK=V_N8 149376
esav TALK=V_N9 150912
esav TALK=V_MAMP 153344
esav TALK=V_VOLTS 157056
```

Here are the battery reporting verbalizations.

The filenames **must** appear as shown as the filenames are fixed in the zNEO code.

Listing 19.12: talk_73181_2025_TREK.fox_44

```
esav TALK=V_F144 160128
esav TALK=V_F145 164864
esav TALK=V_F200 170880
esav TALK=V_F225 176640
esav TALK=V_F250 182656
esav TALK=V_F275 189056
esav TALK=V_F300 194944
esav TALK=V_F325 200832
esav TALK=V_F350 205568
esav TALK=V_F375 210176
esav TALK=CHIRP_UP 214656
esav TALK=CHIRP_DN 216704
esav TALK=CHIRP_UPDN 218752
esav TALK=FD_W0JV 222848
```

Here are the frequency reporting verbalizations. Use the same pattern to add more if you operate on frequencies not listed.

The filenames **must** appear as shown as the filenames are fixed in the zNEO code.

Lines 65 through 67 are the audio emulation of a RADAR chirp.

CHIRP_UP	Ascending frequency audio tone
CHIRP_DN	Descending frequency audio tone
CHIRP_UPDN	Both of the above

Listing 19.13: talk_73181_2025_1.fox_58

```
esav TALK=FD_FOX 272512                                    58
esav TALK=FD_GAZELLE 285824                                59
esav TALK=FD_CATCH 295296                                  60
esav TALK=FD_TUNA 304384                                   61
esav TALK=FD_SILLY_8K 314624                               62
esav TALK=TS1_LA 326016                                    63
esav TALK=TS1R_LA 335872                                   64
esav TALK=TS2_NY 347520                                    65
esav TALK=TS2R_NY 356864                                   66
esav TALK=TS3_SING 367616                                  67
esav TALK=TS3R_SING 376960                                 68
esav TALK=TS4_BOSTON 386688                                69
```

And now we get on with the silly business.

Lines 58 through 62 are nonsense utterances:

FD_FOX	I am a Fox
FD_GAZELLE	I am a Gazelle
FD_CATCH	Catch me if you can
FD_TUNA	I am a tuna-fish sandwich
FD_SILLY_8K	This is getting too silly

Lines 63 through 69 reproduce an old 3-stooges sketch:

TS1_LA	You are now in Los Angeles
TS1R_LA	I am now in Los Angeles
TS2_NY	You are now in New York
TS2R_NY	I am now in New York
TS3_SING	You are now in Sing Sing
TS3R_SING	I am now in Sing Sing
TS1_BOSTON	You are now in Boston

Listing 19.14: talk_73181_2025_1.fox_70

```
esav TALK=2K1_H_9000 395264                                70
esav TALK=2K1_GD_EVE 415616                                71
esav TALK=2K1_CHESS2 427264                                72
esav TALK=2K1_ENJOYA 439552                                73
esav TALK=2K1_JUST_MOM 447616                              74
esav TALK=2K1_MSG_4_U 471168                               75
esav TALK=2K1_MSG_REP 476544                               76
esav TALK=2K1_IGNIT 486400                                 77
esav TALK=2K1_DANGER 494208                                78
esav TALK=2K1_FOOLPROOF 502400                             79
esav TALK=2K1_HUMAN_ERR 529408                             80
```

Listing 19.15: talk_73181_2025_1.fox_81

```
esav TALK=TREK_ABSORPT 556032 81
esav TALK=TREK_AYE_SIR 576640 82
esav TALK=TREK_ENERGY 587392 83
esav TALK=TREK_GREETIN 622848 84
esav TALK=TREK_HAILING 634752 85
esav TALK=TREK_MCCOY_ 642432 86
esav TALK=TREK_QUESTION 655360 87
esav TALK=TREK_SQRE_NOST 692096 88
esav TALK=TREK_SQRE_UNUS 707072 89
esav TALK=TREK_YELLOW 726528 90
esav TALK=SHRK_WDY_CLP 754944 91
```

Line 91 is from Toy Story:

SHRK_WDY_CLP	Look, I'm Woody! Howdy, Howdy, Howdy.

Listing 19.16: talk_73181_2025_1.fox_92

```
esav TALK=HEY_LARRY 766208 92
esav TALK=HEY_MOE 769920 93
esav TALK=SORRY_MOE 772864 94
esav TALK=CURLY_THINKS 780544 95
esav TALK=BIG_IDEA 788736 96
esav TALK=3S_HEY_MOE1 793472 97
esav TALK=3S_HEY_MOE2 804480 98
esav TALK=3S_HEY_MOE3 828416 99
esav TALK=3S_HEY_MOE4 838400 100
esav TALK=3S_PAUSE 861696 101
esav TALK=3S_SHUT_UP 887040 102
esav TALK=3S_TAXIDERMIST 893568 103
esav TALK=3S_TOUPEE 929664 104
esav TALK=3S_VICTIM1 939008 105
esav TALK=3S_WISE_GUYA 945024 106
esav TALK=CLNK_NOISE 950656 107
```

Lines 92 through 106 are 3-stooges clips.

HEY_LARRY	Hey Larry (Curly)
HEY_MOE	Hey Moe (Curly)
SORRY_MOE	Sorry Moe (Larry)
CURLY_THINKS	I'm tryin' to think but nothin' happens (Curly)
BIG_IDEA	What's the big idea!? (Moe)

19.3 FOX Frequency Table

This is a supplemental frequency table.
It is used in this example to add several frequencies not found in the internal table.

Listing 19.17: si5351_frq_cmds_10.fox

```
REM- Call Line: "./si5351a_calc -T2 -F144.100,144.355,5.0 -o-10 -
    ↪ esi5351_frq_cmds1_10.fox "
REM- Output File: si5351_frq_cmds1_10.fox
REM- SI5351 Xtal: 20.000MHz
REM- Freq Offset: -10.000KHz
esav ID=FT,si5351_frq_cmds1_10.fox
esav 144.FOFF -10.000
esav 144.100=139D,0DAC0,F4240
esav 144.105=139D,3C8BF,F4240
esav 144.110=139D,6B6C0,F4240
esav 144.115=139D,9A4C0,F4240
esav 144.120=139D,C92BF,F4240
esav 144.125=139E,03E7F,F4240
esav 144.130=139E,32C80,F4240
esav 144.135=139E,61A7F,F4240
esav 144.140=139E,90880,F4240
esav 144.145=139E,BF680,F4240
esav 144.150=139E,EE47F,F4240
esav 144.155=139F,2903F,F4240
esav 144.160=139F,57E3F,F4240
esav 144.165=139F,86C3F,F4240
esav 144.170=139F,B5A40,F4240
esav 144.175=139F,E483F,F4240
esav 144.180=13A0,1F3FF,F4240
esav 144.185=13A0,4E1FF,F4240
esav 144.190=13A0,7CFFF,F4240
esav 144.195=13A0,ABDFF,F4240
esav 144.200=13A0,DAC00,F4240
esav 144.205=13A1,157BF,F4240
esav 144.210=13A1,445BF,F4240
esav 144.215=13A1,733BF,F4240
esav 144.220=13A1,A21BF,F4240
esav 144.225=13A1,D0FBF,F4240
esav 144.230=13A2,0BB80,F4240
esav 144.235=13A2,3A97F,F4240
esav 144.240=13A2,6977F,F4240
esav 144.245=13A2,9857F,F4240
esav 144.250=13A2,C737F,F4240
esav 144.255=13A3,01F3F,F4240
esav 144.260=13A3,30D40,F4240
esav 144.265=13A3,5FB3F,F4240
```

```
esav 144.270=13A3,8E93F,F4240 41
esav 144.275=13A3,BD73F,F4240 42
esav 144.280=13A3,EC53F,F4240 43
esav 144.285=13A4,270FF,F4240 44
esav 144.290=13A4,55F00,F4240 45
esav 144.295=13A4,84CFF,F4240 46
esav 144.300=13A4,B3AFF,F4240 47
esav 144.305=13A4,E28FF,F4240 48
esav 144.310=13A5,1D4BF,F4240 49
esav 144.315=13A5,4C2BF,F4240 50
esav 144.320=13A5,7B0BF,F4240 51
esav 144.325=13A5,A9EBF,F4240 52
esav 144.330=13A5,D8CBF,F4240 53
esav 144.335=13A6,1387F,F4240 54
esav 144.340=13A6,4267F,F4240 55
esav 144.345=13A6,7147F,F4240 56
```

This example is generated for the V3.72 revision where the internal frequency table is generate without an offset. Each transmitter is characterized and then the appropriate external table is loaded to operate at the correct frequency.

The current offset can be found using the **STAT** command or using the **EDMP 144** command.

This particular table is aimed at the SI5351. This adds some frequencies used by the ICS525 to allow these newer units to be used with the older ones.

You may also note that the **ID=** construct is creeping in to more places to document the files that were used to build the load. Line 5 (**ID=FT**) indicates the source file for this section of the external frequency table.

As more manual work is removed, we make use of the **ID=FT** to track how we constructed the table, frequency offsets in particular.

19.4 FOX Frequency Table

This is a supplemental frequency table for operating in the FM band.

Listing 19.18: si5351_frq_cmds99_0.fox

```
REM- Call Line: "./si5351a_calc -T2 -F87.5,92.95,200.0 -20 -o0 -
    ↪ esi5351_frq_cmds99_0.fox "
REM- Output File: si5351_frq_cmds99_0.fox
REM- SI5351 Xtal: 20.000MHz
REM- Freq Offset: 0.000KHz
esav ID=FT,si5351_frq_cmds99_0.fox
esav 87.500=0F80,00000,F4240,200
esav 87.700=0F8A,3A97F,F4240,200
esav 87.900=0F94,752FF,F4240,200
esav 88.100=0F9E,AFC80,F4240,200
esav 88.300=0FA8,EA600,F4240,200
esav 88.500=0FB3,30D3F,F4240,200
esav 88.700=0FBD,6B6BF,F4240,200
esav 88.900=0FC7,A6040,F4240,200
esav 89.100=0FD1,E09C0,F4240,200
esav 89.300=0FDC,270FF,F4240,200
esav 89.500=0FE6,61A7F,F4240,200
esav 89.700=0FF0,9C400,F4240,200
esav 89.900=0FFA,D6D80,F4240,200
esav 90.100=1005,1D4BF,F4240,200
esav 90.300=100F,57E3F,F4240,200
esav 90.500=1019,927C0,F4240,200
esav 90.700=1023,CD140,F4240,200
esav 90.900=102E,1387F,F4240,200
esav 91.100=1038,4E1FF,F4240,200
esav 91.300=1042,88B80,F4240,200
esav 91.500=104C,C3500,F4240,200
esav 91.700=1057,09C3F,F4240,200
esav 91.900=1061,445BF,F4240,200
esav 92.100=106B,7EF40,F4240,200
esav 92.300=1075,B98C0,F4240,200
esav 92.500=1080,00000,F4240,200
esav 92.700=108A,3A97F,F4240,200
esav 92.900=1094,752FF,F4240,200
```

This example is generated for the V4.07 revision that allow frequency selection in the FM band. Like the 2M tables, each transmitter is characterized and then an appropriate external table is loaded to operate at the correct frequency.

Note the addition of a 4th. numeric parameter in this table. This range of frequency selections require the second MultiSynth to operate with a divisor or 3.

19.5 FOX2X_S0 Message

This is the **S0** message that is sent out periodically (when the S0= schedule is loaded and enabled).

Listing 19.19: FOX2X_S0

```
#                                                                                1
#       Operating Schedules                                                      2
#       The power draw of the DRA818/SA818 may cause the                         3
#       "BATC" form of the report to take too long, so we                        4
#       ALWAYS use the vocal report.                                             5
#       ALSO We're making use of the <CALL> and <NAME> substitution              6
#              inside the fox transmitter!!!                                     7
#                                                                                8
esav ID=FOX2X_S0.fox Baseline                                                    9
esav S0=BATR                                                                     10
esav S0=CONF FM                                                                  11
esav S0=TONE 1.0                                                                 12
esav S0=CWPM 30,-1,-1,-1,-1                                                      13
esav S0=BEGN                                                                     14
esav S0=TALK <CALL>                                                              15
esav S0=TALK <NAME>                                                              16
esav S0=WAIT 0.5                                                                 17
```

Although the previous commands in the **ANN=** section set audio tone and chipping rate, these are set at the beginning of the **S0** message keeping its sound consistent. 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).

Listing 19.20: FOX2X_S0

```
#esav S0='batvc' V 7.2                                                           18
#esav S0=BATC EV 7.2                                                             19
#esav S0=WAIT 0.5                                                                20
```

We may also verbalize battery status, just like we did in the **ANN=** section (although we do **not** in this example). 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 dead air (with carrier). Our example here give us about 500mS of silence.

Listing 19.21: FOX2X_S0

```
#                                                                    21
#   Fill time so they have a chance of finding me                    22
#                                                                    23
esav S0=TONE 'tone'                                                  24
esav S0=CWPM 25,-1,-1,-1,-1                                          25
esav S0=WAIT 0.15                                                    26
#esav S0=CODE hi hi hi                                               27
esav S0=BATC EV 7.2                                                  28
esav S0=WAIT 0.5                                                     29
esav S0=CODE IOWA CITY                                               30
esav S0=CODE AMATEUR RADIO                                           31
esav S0=CODE CLUB FOXHUNT                                            32
esav S0=CODE F W KENT PARK                                           33
esav S0=CODE MARCH 30 2025                                           34
```

At line 24, 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 (i.e. the station identification) 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.

Listing 19.22: FOX2X_S0

```
#                                                                    35
# Prepare (kinda...) for Signoff                                     36
#    these extr REM- commands can be deleted (EZER)                  37
#    and replaced with more CODE commands to adjust time...          38
#                                                                    39
esav S0=REM-                                                         40
esav S0=REM-                                                         41
esav S0=REM-                                                         42
```

Place three spare records into the S0= file to make life easier if we need to add a command or two.

Listing 19.23: FOX2X_S0

```
#                                                                    43
# Signoff                                                            44
#                                                                    45
esav S0=BATR                                                         46
esav S0=TONE 1.0                                                     47
esav S0=CWPM 30,-1,-1,-1,-1                                          48
esav S0=DONE                                                         49
#                                                                    50
```

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 sign-on message at the beginning.

With the V3.73 update, we added a pair of **BATR** commands (lines 10 and 46) to provide battery data when connected to a host system. These commands don't interfere with normal operation.

Here are the **S1** through **S5** messages.
This is the wildlife tracker mode with interrupted carrier.
This set implements the tracker emulation for concurrent operation. The **S1** through **S5** messages are meant to run concurrently. These assume the TOY clocks have has been set recently (within 24 hours). The *chirp*s within the group should occur one after the other. If the TOY clock has drifted, some of the *chirp*s may be on top of each other.

19.6 FOX2X_S1 Message

To activate the chirp schedule, replace the last command in the **ANN=** file (i.e. the **ANN=RUN0,S0** command) with a similar command to activate the **S1** schedule.

Listing 19.24: FOX2X_S1

```
#                                                                    1
#  Wildlife signature (30 second period)                             2
#       ALSO We're making use of the <CALL> and <NAME> substitution  3
#              inside the fox transmitter!!!                         4
esav ID=FOX2X_S1.fox CHiRP                                           5
esav S1=TONE 1.0                                                     6
esav S1=CWPM 30,-1,-1,-1,-1                                          7
esav S1=BEGN                                                         8
esav S1=TALK <CALL>                                                  9
esav S1=TALK <NAME>                                                  10
esav S1=CONF CW                                                      11
esav S1=CHRP 'chrpfrq' 'chrp1' 0.30 55                               12
esav S1=CONF FM                                                      13
esav S1=TONE 1.0                                                     14
esav S1=CWPM 30,-1,-1,-1,-1                                          15
esav S1=DONE SILENT                                                  16
#                                                                    17
```

The **CONF** command set interruptes AM mode. The carrier is removed between chirps.

We follow with the usual configuration of **CWPM** to set the chipping rate to 30 words per minute, **TONE** to set audio tone for the sign-on message to 1KHz, and **BEGN** to enable the clock generator and send the sign-on message.

We send a pair of **TALK** command to verbally announce the callsign and station name.

The **CHRP** command uses a generic substitution to insert the period and offset into the command. This requires *fox_simple* V3.1.

The substitutions used when building from this example is for a 6 second period with each unit setting up for a different offset.

At the end of the chirp traffic, we set the code generator to the same configuration as what we had at the start of the cycle (so it sounds familiar).

The **DONE** command has the **SILENT** flag to suppress station identification traffic. We are transmitting more-or-less continuously, so the start of the next cycle serves to identify the transmitter.
The station identification message will fall on top of the other transmitters inn the group, so the duration of the chirp is set to 300mSec.

19.7 FOX2X_S2 Message

Here is the **S2** message.

Listing 19.25: FOX2X_KC0JFQ_S2

```
#                                                                    1
#  I am a figment of your imagination!                               2
#                                                                    3
esav ID=FOX2X_S2.fox _2025_TREK.hex                                  4
esav S2=BEGN                                                         5
esav S2=TALK <CALL>                                                  6
esav S2=TALK <NAME>                                                  7
#esav S2=TALK SHRK_WDY_CLP                                           8
esav S2=TALK HEY_LARRY                                               9
esav S2=TALK HEY_MOE                                                 10
esav S2=TALK SORRY_MOE                                               11
esav S2=TALK CURLY_THINKS                                            12
esav S2=TALK BIG_IDEA                                                13
esav S2=TALK 3S_TOUPEE                                               14
esav S2=TALK 3S_WISE_GUYA                                            15
esav S2=TALK 3S_HEY_MOE1                                             16
esav S2=TALK 3S_HEY_MOE2                                             17
esav S2=TALK 3S_HEY_MOE3                                             18
#esav S2=TALK 3S_HEY_MOE4                                            19
esav S2=TALK 3S_VICTIM1                                              20
esav S2=TALK 3S_TAXIDERMIST                                          21
esav S2=TALK 3S_PAUSE                                                22
esav S2=DONE                                                         23
```

This is *The Three Stooges*!

It chatters along for just under one minute.

These are added to accommodate a couple more operating scenarios should we choose to implement them. They appear here as placeholders to keep the *FOX2X_KC0JFQ.fox* from changing as these two sequences evolve (this allows this manual to build correctly when S2 or S3 are updated).

19.8 FOX2X_S3 Message

Here is the **S3** message.

Listing 19.26: FOX2X_S3

```
#
#  I am also a figment of your imagination!
#
esav ID=FOX2X_S3.fox _2025_TREK.hex
#
esav S3=BEGN
esav S3=TALK <CALL>
esav S3=TALK <NAME>
esav S3=TALK 2K1_H_9000
esav S3=TALK 2K1_GD_EVE
esav S3=TALK 2K1_CHESS2
esav S3=TALK 2K1_ENJOYA
esav S3=TALK 2K1_JUST_MOM
esav S3=TALK 2K1_MSG_4_U
esav S3=TALK 2K1_MSG_REP
esav S3=TALK 2K1_IGNIT
esav S3=TALK 2K1_DANGER
esav S3=TALK 2K1_FOOLPROOF
esav S3=TALK 2K1_HUMAN_ERR
esav S3=DONE
```

This is a set of clips from *2001, A Space Odyssey*.

It also runs for just under one minute.

19.9 FOX2X_S4 Message

Here is the **S4** message.

Listing 19.27: FOX2X_KC0JFQ_S4

```
#
#  I am also a figment of your imagination!
#
esav ID=FOX2X_S4.fox _2025_TREK.hex
#
esav S4=BEGN
esav S4=TALK <CALL>
esav S4=TALK <NAME>
esav S4=TALK TREK_YELLOW
esav S4=TALK TREK_AYE_SIR
esav S4=TALK TREK_ABSORPT
#esav S4=TALK TREK_ENERGY
esav S4=TALK TREK_GREETIN
esav S4=TALK TREK_HAILING
esav S4=TALK TREK_MCCOY_
esav S4=TALK TREK_QUESTION
esav S4=TALK TREK_SQRE_NOST
esav S4=TALK TREK_SQRE_UNUS
esav S4=DONE
```

Here we encounter James Kirk and Mr. Spock from *Star Trek*.

It also chatters along for just under one minute.

19.10 FOX2X_S5 Message

Here is the **S5** message.

Listing 19.28: FOX2X_S5

```
#                                                                                1
#  /home/wtr/Radio/halo_term/fox_simple -SFOX2X -c150 -t10 -CW0JV -NFOX21 -Q144  2
    ↪ .225 -R"360,60" -A-7 -fSX_CHIRP.fox
#                                                                                3
#                                                                                4
#================================================                                5
#                                                                                6
esav ID=FOX2X_S5.fox Wildlife                                                    7
esav S5=TONE 1.0                                                                 8
esav S5=CWPM 30,-1,-1,-1,-1                                                      9
esav S5=BEGN                                                                     10
esav S5=TALK <CALL>                                                              11
esav S5=TALK <NAME>                                                              12
#                                                                                13
esav S5=CONF CW                                                                  14
#                                                                                15
#------------------------------------------------                                16
#  Make it do a animal tracker Chirp                                             17
#                                                                                18
esav S5=WAIT 3                                                                   19
#                                                                                20
# 'chrpfrq' frequency (set in  script)                                           21
#   12      second cycle                                                         22
# 'chirp'   second offset into cycle (set in load script)                        23
#   0.100   second tone duration                                                 24
#   26      repeat count (312 seconds)                                           25
#                                                                                26
esav S5=CHRP  'chrpfrq' 12 'chirp_up' 0.100 46                                   27
#                                                                                28
esav S5=WAIT 3                                                                   29
#                                                                                30
#------------------------------------------------                                31
#                                                                                32
esav S5=CONF FM                                                                  33
#                                                                                34
esav S5=TONE 1.0                                                                 35
esav S5=CWPM 30,-1,-1,-1,-1                                                      36
esav S5=DONE                                                                     37
```

A late addition to the fox transmitter sequences, this is a simple emulation of the animal tracker. Here we emit a simple tone on a repeating schedule.

The timing spec is almost identical to the following **S6** message.

19.11 FOX2X_S6 Message

Here is the **S6** message.

Listing 19.29: FOX2X_S6

```
#                                                                                1
#  /home/wtr/Radio/halo_term/fox_simple -SFOX2X -c150 -t10 -CW0JV -NFOX21 -Q144  2
    ↪ .225 -R"360,60" -A-7 -fSX_CHIRP.fox
#                                                                                3
#                                                                                4
#=================================================                               5
#                                                                                6
esav ID=FOX2X_S6.fox CHiRP                                                       7
esav S6=TONE 1.0                                                                 8
esav S6=CWPM 30,-1,-1,-1,-1                                                      9
esav S6=BEGN                                                                     10
esav S6=TALK <CALL>                                                              11
esav S6=TALK <NAME>                                                              12
#                                                                                13
esav S6=CONF CW                                                                  14
#                                                                                15
#-------------------------------------------------                               16
#  Make it do a REAL CHiRP                                                       17
#                                                                                18
esav S6=WAIT 3                                                                   19
#                                                                                20
#   12       second cycle                                                        21
# 'chirp'    second offset into cycle (set in load script)                       22
#   0.05     second  turn-on delay (was tone duration)                           23
#   26       repeat count (312 seconds)                                          24
#                                                                                25
esav S6=CHRP CHIRP_UP 12 'chirp_up' 0.05 23                                      26
esav S6=CHRP CHIRP_DN 12 'chirp_dn' 0.05 23                                      27
#                                                                                28
esav S6=WAIT 3                                                                   29
#                                                                                30
#-------------------------------------------------                               31
#                                                                                32
esav S6=CONF FM                                                                  33
#                                                                                34
esav S6=TONE 1.0                                                                 35
esav S6=CWPM 30,-1,-1,-1,-1                                                      36
esav S6=DONE                                                                     37
```

This is the radar chirp emulation test sequence. As the users manual was edited, the assumed cycle time for this schedule is 600 seconds. the Chirping activity takes 552 seconds, leaving 48 seconds for the sign-on and sign-off messages.

We use the more-or-less standard sign-on and sign-off. Lines 7 through 13 configure and send the callsign and nickname. Lines 32 through 36 send the standard sign-off message in code.

The **S6=CONF -CW** command on line 36 restores the fox transmitter to **not** operate in an interrupted carrier mode. This keeps the sign-on message clearly understandable.

The **S6=CONF CW** command on line 13 configures to operate in an interrupted carrier mode. Much like a code transmitter, we only send a modulated carrier. We suppress the carrier when there is no audio traffic outgoing.

Lines 25 and 26 send the audio clip in place of a fixed tone. In this example the **CHIRP_UP** audio file is an ascending tone from 300Hz to 1500Hz. The **CHIRP_UP** audio file is a descending tone from 1500Hz down to 300Hz.

Each audio clip is 500mSec long, fitting well into the 12 second cycle period. The 12 second cycle is repeated 26 times requiring 312 seconds. This leaves 48 seconds in the 360 second overall cycle (see the *S6=* schedule in listing 19.1.3 on page 321) for the sign-on and sign-off messages.

19.12 FOX2X_S7 Message

Here is the **S7** message.

As with the S1 message, this is activated by altering the last command in the ANN= file.

Listing 19.30: FOX2X_S7

```
#                                                                    1
REM- 0          1          2          3                             2
REM- 01234567890123456789012345678901                               3
esav ID=FOX2X_S7.fox Field Day                                       4
esav S7=CONF FM                                                      5
esav S7=TONE 1.0                                                     6
esav S7=CWPM 30,-1,-1,-1,-1                                          7
esav S7=BEGN                                                         8
esav S7=TALK <CALL>                                                  9
esav S7=TALK <NAME>                                                  10
esav S7=WAIT 1.0                                                     11
esav S7=TALK FD_W0JV                                                 12
esav S7=WAIT 1.0                                                     13
esav S7=CWPM 30,-1,-1,-1,-1                                          14
esav S7=TONE 1.5                                                     15
esav S7=CODE IOWA CITY                                               16
esav S7=CODE AMATEUR RADIO CLUB                                      17
esav S7=CODE FIELD DAY OPERATIONS                                    18
esav S7=CODE AT F W KENT PARK                                        19
esav S7=WAIT 1.0                                                     20
esav S7=TALK FD_GAZELLE                                              21
esav S7=WAIT 1.0                                                     22
esav S7=REM-                                                         23
esav S7=REM-                                                         24
esav S7=REM-                                                         25
#                                                                    26
#                                                                    27
#                                                                    28
# Signoff                                                            29
#                                                                    30
esav S7=TONE 1.0                                                     31
esav S7=CWPM 30,-1,-1,-1,-1                                          32
esav S7=DONE                                                         33
#                                                                    34
```

This is a prototype Field-Day message.

19.13 FOX2X_S8 and FOX2X_S9

Here is the **S8/S9** message pair.

Although similar to activating S0 and S1 message traffic, these messages are meant to be handled by only two transmitters. The S8= message on one and the S9= message on a second unit. The remaining stations in the hunt group would run their S0= or S1= messages normally.

Listing 19.31: FOX2X_KC0JFQ stooge

```
#                                                                    128
#  A bit of levity                                                   129
#                                                                    130
#include 'sx_stooge'                                                 131
```

This is the include line to grab the two sequences for voicing the conversation between Shemp and Moe.

In this set of sequences we try to expose some of the fine timing capabilities buried in the software.

For the older crowd, recall the episode when Shemp hypnotizes Moe. Shemp takes Moe from Los Angeles to New York. And then upstate to Sing-Sing whereupon Moe picks up a straight back chair. Shemp attempts to take Moe on to Boston, but Moe is stuck: "I am now in Sing-Sing".

The sequence uses the expanded function of the **WAIT** command to have one transmitter plays the part of Shemp, sending Moe across the country. A second transmitter plays the part of Moe and answers Shemp.

Listing 19.32: FOX2X_STOOGE_S8

```
#                                                                    4
#==================================                                  5
#                                                                    6
esav ID=FOX2X_STOOGE.fox S8 Shemp                                    7
esav S8=TONE 1.0                                                     8
esav S8=CWPM 30,-1,-1,-1,-1                                          9
esav S8=CONF FM                                                      10
esav S8=BEGN                                                         11
esav S8=TALK <CALL>                                                  12
esav S8=TALK <NAME>                                                  13
esav S8=CODE SING SING                                               14
esav S8=CONF CW                                                      15
esav S8=CODE E                                                       16
#                                                                    17
esav S8=WAIT 'stooge'/40                                             18
esav S8=TALK TS1_LA                                                  19
esav S8=WAIT 'stooge'/50                                             20
esav S8=TALK TS2_NY                                                  21
esav S8=WAIT 'stooge'/60                                             22
esav S8=TALK TS3_SING                                                23
esav S8=WAIT 'stooge'/70                                             24
esav S8=TALK TS4_BOSTON                                              25
#                                                                    26
esav S8=WAIT 'stooge'/80                                             27
esav S8=CONF FM                                                      28
esav S8=TONE 1.2                                                     29
esav S8=CODE BEEN LISTENIN TO                                        30
esav S8=CODE THE THREE STOOGES                                       31
#                                                                    32
esav S8=TONE 1.0                                                     33
esav S8=CWPM 30,-1,-1,-1,-1                                          34
esav S8=DONE                                                         35
```

See the worksheets in section 22.4 on page 376. A detailed timing layout for this **S8=** is found in section 22.5 on page 377.

After our normal sign-on message traffic, we use an enhancement to the **WAIT** command to synchronize with the clock in the same manner as the **MODS** scheduling.

When the timing argument to **WAIT** contains a slash ("/") a second numeric argument is expected. The **WAIT** command arguments, then, are the scheduling period and the offset into the period. The **WAIT** command waits for that scheduling point before proceeding.
IN the **S8=** example here, the 'stooge' parameter is replaced by a value (as loaded in the test-bed, I used 120 seconds) as the commands are loaded into the fox transmitter by the **fox_simple** utility.

This is the other half of the conversation.

Listing 19.33: FOX2X_STOOGE_S9

```
#
esav ID=FOX2X_STOOGE.fox S9 Moe
esav S9=TONE 1.0
esav S9=CWPM 30,-1,-1,-1,-1
esav S9=CONF FM
esav S9=BEGN
esav S9=TALK <CALL>
esav S9=TALK <NAME>
esav S9=CODE SING SING
esav S9=CONF CW
esav S9=CODE E
#
esav S9=WAIT 'stooge'/44
esav S9=TALK TS1R_LA
esav S9=WAIT 'stooge'/54
esav S9=TALK TS2R_NY
esav S9=WAIT 'stooge'/64
esav S9=TALK TS3R_SING
esav S9=WAIT 'stooge'/74
esav S9=TALK TS3R_SING
#
esav S9=WAIT 'stooge'/100
esav S9=CONF FM
esav S9=TONE 1.6
esav S9=CODE BEEN LISTENIN TO
esav S9=CODE THE THREE STOOGES
#
esav S9=TONE 1.0
esav S9=CWPM 30,-1,-1,-1,-1
esav S9=DONE
```

In the **S8=TALK**, we ask the question and in the **S9=TALK**, we answer. Note that the **S8=WAIT** that control timing is scheduled 4 seconds before the **S9=WAIT**.

This sign-on messages are scheduled by the **MODS** command and then subsequent timing is controlled by the **WAIT** command. The timing control method in the fox transmitter allows for the required granularity and precision.

The other command that enables this behavior is the **CONF CW** command. This configures the RF control in a manner that unkeys the transmitter when there is not traffic being sent. We are manipulating the control line that drives the PTT pin on the DRA818/SA818 hardware. On the CHiRP amplifiers, this control line is connected to the on-board power switch which makes the CHiRP amplifier act like the DRA818/SA818 hardware.

This feature will only work with the newer CHiRP amplifiers and the DRA818/SA818 hardware. Other amplifiers lack the compatible switching logic.

Near the end, we switch back to more-or-less normal FM mode, where the carrier is constantly on with the **CONF -AM** command. This allows the sign-off message (**DONE**) to be sent without having the hunters hear their radios dropping when carrier is lost.

19.14 S_MOxx.fox and S_sprint.fox

Here are fragments from the **IARU** message sequence.

The 5 sequences all look the same with only the 3 character message changing for each transmitter. The 3-letter groups are MOE MOI MOS MOH MO5.
M is 2-dash, O is 3-dash and the E,I,S,H, and 5 are one through 5 dits.
So we hear "*dah dah dah dah dah dit*" for the first, "*dah dah dah dah dah dit dit*" for the second, and so-on.

19.14.1 S_MOxx.fox

This is the standard *one minute* message.

Listing 19.34: S_MOxx

```
esav ID=S_MOxx.fox S1,MOE                                    1
esav S1=TONE 1.4                                             2
esav S1=CWPM 20,-1,-1,-1,-1                                  3
esav S1=BEGN                                                 4
esav S1=WAIT 1.0                                             5
esav S1=CWPM 15,1,3,15,14                                    6
esav S1=CODE,MOE,MOE,MOE,MOE                                 7
esav S1=CODE,MOE,MOE,MOE,MOE                                 8
esav S1=CODE,MOE,MOE                                         9
esav S1=WAIT 1.0                                             10
esav S1=CWPM 20,-1,-1,-1,-1                                  11
esav S1=DONE                                                 12
```

Each of the messages in this set takes 55.350 seconds to send. This comfortably fits in the 60 second window.

On line 6 the time after each 3-letter word is stretched out (i.e. the inter-word time) to keep the time required to send the group equal across all 5 transmitters. This hack keeps the message time consistent for all transmitters.

The same line for all of the transmitters appear as:

```
esav S1=CWPM 15,1,3,15,14                                    6
```

```
esav S2=CWPM 15,1,3,13,14                                    19
```

```
esav S3=CWPM 15,1,3,11,14                                    32
```

```
esav S4=CWPM 15,1,3,9,14                                     45
```

```
esav S5=CWPM 15,1,3,7,14                                     58
```

Note that the 4th. parameter changes from 15 to 7 (delta of 2 for each transmitter). This change serves to keep the time required to send the last letter of the 3-letter group the same in spite of the changing word length.

19.14.2 S_sprint.fox

This is typical of the sprint messages.

A sprint only allocates 12 seconds to each station in a one minute cycle. The TOY clock needs to be recently updated and track well for this operation to avoid stepping on other transmitters!

Both the **BEGN** and **DONE** have the **SILENT** modifier to suppress the typical signon as this would take far too long to send.

Listing 19.35: S_sprint

```
esav ID=S_sprint.fox S1,MOE                                    1
esav S1=TONE 1.4                                               2
esav S1=CWPM 20,-1,-1,-1,-1                                    3
esav S1=BEGN SILENT                                            4
esav S1=CODE,<CALL>,                                           5
esav S1=CWPM 15,-1,-1,-1,-1                                    6
esav S1=CODE,MOE,MOE,MOE                                       7
esav S1=DONE SILENT                                            8
```

We only provide the station callsign using the **CODE,<CALL>** to send out the stored station callsign. The station nickname is **not** sent. The five variations (S1= through S5=) change the 3 -letter sequence in line 7 in the same manner as the S_MOxx.fox sequences.

This sequence hasn't yet been timed! Will probably have the **CWPM** command adjusted and the number of *MOE* repetitions. Also keep in mind that the station callsign consumes air time. Adjustments, specific to your callsign, will need to be made.

Do note that **CODE <CALL>** will not be properly substituted with the actual callsign and nickname prior to version V4.09.
The earlier versions will send the **<CALL>** text without being translated. In this example, you would be out of compliance with the rules when running the earlier software.

19.15 fox20.sh

This is the shell script used to load the ICARC fox transmitters. The single argument to the shell script is the transmitter nunber. It is reformed into the transmitter nickname on line 36.

Listing 19.36: fox20.sh setup

```
#!/usr/bin/bash
#
#
FOXS=/home/wtr/Radio/halo_term/fox_simple
#FOXS=echo
#
#  variables, unique to our fox hunt
#
CALL=KC0JFQ
CALL=W0JV
FREQ=144.225
OFF=300,0
RUN=600,0
STOOGE=120
CHRP1="6,0"
TACH=-SFOX2X
TACH=" "
FOXCMD=FOX2X_KC0JFQ.fox
OFF_FILE=""
FM_FILE=""
IARU_80M=""
SCHED=S0
CHRPFRQ=1.0
CHIRP_UP=0
TONE=1.0
LOG=FOX2X_KC0JFQ
TALK_FILE=talk_73181_2025_1.fox
SYNTH_DEV=SI5351
SYNTH_SET1=8MA
SYNTH_SET2=CLK0
SX_STOOGE=FOX2X_STOOGE.fox
SPARE1=not
SPARE2=used
BATVC=BATV
#
#  variables, unique to individual fox tranmsitters
```

Default values are assigned for all of the variables.

19.15.1 fox20.sh FOX5

This station is one of the older 102-73161-25 boards. I uses the ICS525 frequency synthesizer.

Listing 19.37: fox20.sh FOX5

```
#=============================================================== 58
#                                                               59
#  FOX5                                                         60
#                                                               61
if [ "$1" = "5" ]; then                                         62
    TACH=-SFOX5                                                 63
    TACH=" "                                                    64
    FREQ=144.285                                                65
    BATVC=BATC                                                  66
    SCHED=S0                                                    67
    OFF=360,0          #    FOX5       144.285                 68
    CHRP1="6,3"                                                 69
    CHRPFRQ=1.3                                                 70
    SYNTH_DEV="ICS525"                                          71
    SYNTH_SET1=BMON                                             72
    SYNTH_SET2=73161                                            73
    OFF_FILE="/home/wtr/Fox_Tx_73181/trunk/ics525_table.fox"    74
    SX_STOOGE="freq_null.fox"                                   75
    TALK_FILE="talk_fox5_rxxk.fox"                              76
    SPARE1=VOICE                                                77
```

In lines 68 through 70 ("SYNTH_" lines) we change the configuration to accommodate the 102-73161-25 hardware. In lines 71 through 73 we deal with the lack of FLASH memory by leaving out the SI5351 frequency table and the full TALK file (talk_fox5_2025_1 has only the station callsign and our nickname).

19.15.2 fox20.sh FOX20

The FOX20 station hardware is the 102-73181-5 board. This uses the (obsolete) 80-pin zNEO package.

Listing 19.38: fox20.sh FOX20

```
#-------------------------------------------------------------------------
#
#  FOX20
#                FREQ=144.285  As part of a hunt group
#                FREQ=144.300  As a training transmitter
#
if [ "$1" = "20" ]; then
    FREQ=144.285
    FREQ=144.300
    SCHED=S0
    OFF=60,0          #   FOX20    144.300
    OFF_FILE="/home/wtr/Fox_Tx_73181/trunk/si5351_frq_cmds_10.fox"
    FM_FILE="/home/wtr/Fox_Tx_73181/trunk/si5351_frq_cmds99_5.fox"
    TALK_FILE=talk_73181_2025_TREK_2.fox
    LOG=FOX20_KC0JFQ
fi
```

We select the operating frequency for the first group, same as FOX5, but use a different scheduling offset.

You may notice that this transmitter group is not on a 25KHz step. This is a limitation of the ICS525 synthesizer used in the older transmitters in the group. The 102-73181-5 hardware deals well with this, but will not voice the frequency in the **ANN=** message.

19.15.3 fox20.sh FOX21

The FOX21 station hardware is the 102-73181-10 board.

Listing 19.39: fox20.sh FOX21

```
#=============================================================== 189
#                                                               190
#  FOX21 and FOX22 can be operated in the                       191
#    "3 Stooges" 'SING SING' mode...                            192
#                                                               193
if [ "$1" = "21" ]; then                                        194
    SCHED=S6                                                    195
    OFF=360,0        #   FOX21     144.225                      196
    RUN=600,0                                                   197
    CHRP1="6,0"                                                 198
    CHRPFRQ=1.0                                                 199
    CHIRP_UP=0                                                  200
    TONE=1.3                                                    201
    LOG=FOX21_KC0JFQ                                            202
#NO# TALK_FILE=talk_73181_2025_TREK_2.fox FLASH too small!      203
    OFF_FILE="/home/wtr/Fox_Tx_73181/trunk/si5351_frq_cmds_8.fox"  204
    FM_FILE="/home/wtr/Fox_Tx_73181/trunk/si5351_frq_cmds99_4.fox"  205
    IARU_80M="/home/wtr/Fox_Tx_73181/trunk/si5351_frq_cmds80_1.fox"  206
fi                                                              207
```

This is the first station in the second hunt group. Our frequency is set clear up in line 11 (i.e. the default of 155.225MHz). The SI5351 crystal appears to have an 8KHz offset, so we load the table (freq_5351-08.fox) the corrects for this error.

We also see that this transmitter has a small FLASH device so we can't store the extended audio file system.

19.15.4 fox20.sh FOX27

The FOX27 station hardware is the 102-73181-10 board.

Listing 19.40: fox20.sh FOX27

```
#=============================================================== 296
# 297
if [ "$1" = "27" ]; then 298
    OFF=360,30      #   FOX27     144.325 299
    CHRP1="6,0" 300
    CHIRP_UP=0 301
    FREQ=144.325 302
    LOG=FOX27_KC0JFQ 303
    TALK_FILE=talk_73181_2025_TREK_2.fox 304
    OFF_FILE="/home/wtr/Fox_Tx_73181/trunk/si5351_frq_cmds_12.fox" 305
    FM_FILE="/home/wtr/Fox_Tx_73181/trunk/si5351_frq_cmds99_6.fox" 306
    TONE=1.3 307
fi 308
```

FOX21 and later all use the 102-73181-10 board revision, so we match FOX21 in that respect. All we change is out operating frequency (to 144.325MHz in line 275) and operating schedule (in line 272).

Also note that this unit is calling for the extended audio file system to be loaded in line 284 as there is a large FLASH device on this board.

19.15.5 fox20.sh fox_simple

Now that we have all the variable set for our station, we can invoke the *fox_simple* loader to load up the FRAM with out sequence.

Listing 19.41: fox20.sh fox_simple

```
460 $FOXS $TACH -c150 -t10 -f$FOXCMD -l$LOG.log \
461         -Xchrp1=$CHRP1 \
462         -Xchrpfrq=$CHRPFRQ \
463         -Xsched=$SCHED \
464         -Xtone=$TONE \
465         -Xstooge=$STOOGE \
466         -Xchirp_up=$CHIRP_UP \
467         -Xchirp_dn=$CHIRP_DN \
468         -Xrun=$RUN \
469         -Xruns6=$RUN \
470         -Xsynth_dev=$SYNTH_DEV \
471         -Xsynth_set1=$SYNTH_SET1 \
472         -Xsynth_set2=$SYNTH_SET2 \
473         -Xtalk_file=$TALK_FILE \
474         -Xsx_stooge=$SX_STOOGE \
475         -Xspare1=$SPARE1 \
476         -Xspare2=$SPARE2 \
477         -Xbatvc=$BATVC \
478         -Xoff_file=$OFF_FILE \
479         -Xfm_file=$FM_FILE \
480         -Xiaru_80m=$IARU_80M \
481         -C$CALL -N$NAME -R$OFF -Q$FREQ
```

Looking way back at line 4, we see the location of the *fox_simple* utility image.

The **$TACH** variable (from lines 16 and 60) gives us the USB device that connects us to the target station.

We ask for a 150mS delay between records (-c150) and to load the clock with a truncated time of modulo 10 days. The **$FOXCMD** variable points to the master fox command file. And we save a log of the sent commands in the **$LOG.log** file.

All of the **-x** lines setup parameter substitutions in the **$FOXCMD** file. Looking through the FOX2X_KC0JFQ.fox records above, you will find where all these generic substitutions occur.

The last line has the fixed substitutions.

- **-C$CALL** for the station callsign
- **-N$NAME** for the station nickname
- **-R$OFF** for the station scheduling (period and offset)
- **-Q$FREQ** for the operating frequency
- **-A$OFF_FILE** for the SI5351 external frequency table

fox_binary

We can also use *fox20.sh* to generate a log file with the translated commands and use the *fox_binary* utility to perform a fast load of the transmitter FRAM.

Leaving the **$TACH** variable off of the command line will suppress the serial channel selection while still generating the translated log for the *fox_binary* utility.

19.16 ONCE Testing

The **ONCE** command is used to time the message as it is sent out. We see the command reports as the sequence runs. This operation is used to time the message so we can adjust to fit as needed.

```
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
```

The *sts00,13* Execution Time:* is 53.8 seconds.

Given five stations on a 5 minute cycle, we allocate 60 seconds to each station. This test indicates we have about 6 seconds to spare, which is a reasonable to account for the variation in the TOY clock between stations.

We can expect the stations to not overlap each-other during the hunt.

19.17 ICARC Fox Hunt Configuration

This section talks about the scripts in section 19.1 on page 317 as used to implement the W0JV fox hunts. Although there are six sequences stored in these FRAM of the fox transmitters, only three will be discussed here. Using the remaining three is much the same as that discussed here.

Note that the FLASH is loaded with audio clips tailored for the local club and authors callsigns. These clips will not be used outside the ICARC unit so you will need to load your own callsign into FLASH if you load the authors audio files. The number of callsigns that may be stored in FLASH are limited only by the size of the FLASH (i.e. you may store multiple callsigns if you need them). In our working example, the **W0JV** and **KC0JFQ** clips are defined in lines 20 and 21 of the talk_73181_2025_1.fox file shown in section 19.2 on page 327.

We also have vocalizations for 19 unique transmitter nicknames, FOX20 through FOX38, allowing for three 6-unit groups with this audio load. These names, being generic, may be reused as they exist.

You are free, of course, to regenerate all of the voice fragments. The voice fragments on lines 1 through 19 are used to vocalize a battery report (using the **BATV** command). The voice fragments on lines 35 through 44 are used to vocalize the operating frequency in the announce message shown in section 19.1.6 on page 324 on lines 100 and 101.

Keep in mind that the frequency table in section on page 332 is one of about 20 that were generated for the authors transmitter groups. The fox transmitters built by the author all require slightly different offsets to place the carrier at the center of the receive window. None of these transmitters have frequency trim parts on the RF clock generator.

You must measure the frequency offset on your units and (individually) select accordingly.

The **INI=** section that is loaded into the FRAM in the fox transmitter is uniquely tailored for each transmitter as it is loaded.

The **name** and **call** (lines 47 and 48 in section 19.1.3 on page 321) will be changed to our callsign (for our club hunts to **W0JV**) and the nickname for each fox (FOX21..FOX32 for two 6-unit hunt groups).

Down at lines 61 through 70 we define up to ten schedules. We will only activate one of them, but having them all stored in FRAM allows us to easily change the personality to fit the particular style of hunt we will be operating.

For our typical hunt, we operate 6-unit groups on a 6 minute cycle. Each transmitter is allocated a one minute window in the cycle. The **S0=** schedule is loaded as 360,0 in the first unit, 360,60 in the second and sliding each successive unit by 60 seconds. The second group operates on a different frequency. Should you operate with a third, simply select another unique operating frequency.

The **ANN=** section that is loaded into the FRAM is also tailored to each individual transmitter. The callsign and nickname (section 19.1.6 on page 324) on lines 94 and 95 are handled within the fox transmitter. The working callsign and nickname having been saved away in the **INI=** section earlier. The **<CALL>**

We assign the operating frequency as the FRAM is loaded. That frequency is verbalized in lines 100 and 101 and actually set in line 105. This new frequency assignment will take effect at the next **BEGN** command, so we don't *pull the rug out from under ourselves.*

The last step is to activate one of the available schedules. Line 107, like most of the rest of the unit specific setup, is tailored to the particular unit when FRAM is loaded using a shell script.

19.17.1 ICARC S0= Sequence

The normal fox hunt is implemented using the **S0=** sequence shown in section 19.5 on page 334.

This sequence is configured to be used for battery life evaluation (the **BATR** commands) without impacting hunt operations. One **BATR** command with the RF amplifier disabled (line 119) and one with it enabled (line 155).

As expected, we take callsign and nickname from values stored in the **INI=** sequence. As we have the verbalization stored in FLASH, we can verbally identify the transmitter (lines 124 and 125). The callsign will have been sent in code wit the **BEGN** command (line 123). Note that the sign-on message (i.e. the **BEGN** command) and the sign-off message (i.e. the **BEGN** command) are sent wit the same chipping rate and at the same audio frequency. This makes the start and stop sound the same on all units to allow the hunter to recognize when we are about to switch over to a new transmitter.

The main body of the message sounds a bit different from each transmitter as we define the audio frequency in the shell script as we load that from FRAM.

The hunter should be able to recognize that we've recycled back to the first unit in the group by the drop in frequency of the main message.

The sign-off message (i.e. the **BEGN** command), as mentioned previously, is set with a different chipping rate (i.e. back to that of the sign-on message) and audio tone frequency (again, back to that of the sign-on message).

Reverting to the sound of the sign-on message, as mentioned, is intended to suggest to the hunter that the transmitter is about to go quiet.

19.17.2 ICARC S1= Sequence

An alternative hunt, which is **far more difficult** to hunt is embodied in the **S1=** sequence (in section 19.6 on page 336).

This sequence requires that the TOY clock be set the day prior to the hunt so that the TOY clock hasn't drifted. We use the **CHRP** command to send a short audio burst (300 mSec is specified in the command arguments). The sign-on message occurs using the normal scheduling arguments, that is every 360 seconds with each transmitter shifted by an additiolnal 60 seconds.

Once operating, the short audio tone is sent every six seconds. The six units are staggered by one second, so you get a short *peep* from a different unit each second. Very hard to locate!

As with all the other fragments discussed here, the **'chrpfrq'** and **'chrp1'** are substituted when running the shell script to load the FRAM. The first unit get loaded with *1.3,6,3* and the next with *1.0,6,0* such that each unit is given the same period (2nd. argument) and a unique frequency (1st. argument) and offset (3rd. argument).

19.17.3 ICARC S6= Sequence

The **S6=** sequence operates in a similar fashion while making use of a feature added in the V3.85 software. Here we change from using a simple tone from the CW tone generator to using a short audio clip. The specific clips used are of a rising audio tone and a falling audio tone. An audio emulation, if you will, of a RADAR chirp.

We use the **CHRP** command with a filename rather than an audio tone value. As long as the audio filename starts with a non-numeric character, the software will attemp a lookup in the talk directory.

The remaining arguments to the **CHRP** command perform the same job with the exception of the argument that specifies tone duration. The function of that argument changes to providing an RF settling time before audio data begins flowing.

The cycle time is expanded to 12 seconds to relax demands on the TOY clock and to accommodate the slightly longer time that the RF amplifier is enabled. Each of the six units in the group are staggered by 2 seconds.

For both the **S1=** sequence and the **S6=** sequence we alter the bit in the configuration array that disables the RF amplifier when the fox is not actively transmitting (this is what the **CONF -AM** command does). In this configuraton, the RF amplifier is only powered when sending audio. When sending code, the time between dits and dahs have the RF amplifier powered down.

19.18 FOX21_KC0JFQ.log

The log file lists the commands sent to the target fox system.

```
000 esav REM= ./fox_simple V3.6 Apr 20 2025
001 esav REM= -S Undefined
002 esav REM= -F FOX2X_KC0JFQ.fox
003 esav REM= 2025-06-16T19:47:02
004 esav ID=FL talk_73181_2025_1.fox
005 esav ID=FL 2025-06-03T12:19:13
006 esav TALK=KC0JFQ 0
007 esav TALK=W0JV 5760
008 esav TALK=FOX20 11904
009 esav TALK=FOX21 14848
010 esav TALK=FOX22 19072
011 esav TALK=FOX23 23424
012 esav TALK=FOX24 28672
013 esav TALK=FOX25 33408
014 esav TALK=FOX26 39552
015 esav TALK=FOX27 44544
016 esav TALK=FOX28 49152
017 esav TALK=FOX29 53632
018 esav TALK=FOX30 58624
019 esav TALK=FOX31 62464
020 esav TALK=FOX32 67200
021 esav TALK=FOX33 72064
022 esav TALK=FOX34 77696
023 esav TALK=FOX35 83328
024 esav TALK=FOX36 88832
025 esav TALK=FOX37 94208
026 esav TALK=FOX38 99328
027 esav TALK=FOX39 103552
028 esav TALK=BATTI 108928
029 esav TALK=BATTV 113152
030 esav TALK=REG5 117632
031 esav TALK=POINT 122752
032 esav TALK=V_HZ 124160
033 esav TALK=V_KHZ 126592
034 esav TALK=V_MHZ 129792
035 esav TALK=V_N0 132992
036 esav TALK=V_N1 135680
037 esav TALK=V_N2 137472
038 esav TALK=V_N3 139648
039 esav TALK=V_N4 141568
040 esav TALK=V_N5 143488
041 esav TALK=V_N6 145664
042 esav TALK=V_N7 147456
043 esav TALK=V_N8 149376
044 esav TALK=V_N9 150912
045 esav TALK=V_MAMP 153344
046 esav TALK=V_VOLTS 157056
047 esav TALK=V_SEC 160128
048 esav TALK=V_TIRP 163328
049 esav TALK=V_F144 167424
050 esav TALK=V_F145 172160
051 esav TALK=V_F200 178176
052 esav TALK=V_F225 183936
053 esav TALK=V_F250 189952
054 esav TALK=V_F275 196352
055 esav TALK=V_F300 202240
056 esav TALK=V_F325 208128
057 esav TALK=V_F350 212864
058 esav TALK=V_F375 217472
059 esav TALK=CHIRP_UP 221952
060 esav TALK=CHIRP_DN 224000
061 esav TALK=CHIRP_UPDN 226048
062 esav TALK=FD_W0JV 230144
063 esav TALK=FD_FOX 279808
064 esav TALK=FD_GAZELLE 293120
065 esav TALK=FD_CATCH 302592
066 esav TALK=FD_TUNA 311680
067 esav TALK=FD_SILLY_8K 321920
068 esav TALK=TS1_LA 333312
069 esav TALK=TS1R_LA 343168
070 esav TALK=TS2_NY 354816
071 esav TALK=TS2R_NY 364160
072 esav TALK=TS3_SING 374912
073 esav TALK=TS3R_SING 384256
074 esav TALK=TS4_BOSTON 393984
075 esav TALK=SHRK_WDY_CLP 402560
076 esav TALK=HEY_LARRY 413824
077 esav TALK=HEY_MOE 417536
078 esav TALK=SORRY_MOE 420480
079 esav TALK=CURLY_THINKS 428160
080 esav TALK=BIG_IDEA 436352
081 esav TALK=CLNK_NOISE 441088
082 esav TALK=CLUNKING 445952
083 esav TALK=CLUNKX3 449280
084 esav TALK=CLUTCH 457472
085 esav ID=FL SIZE 0x72180
086 esav INI=TIME
087 esav INI=WAIT 0.5
088 esav INI=TIME
089 esav INI=EPOC -5.0
```

```
090 esav INI=NAME FOX21
091 esav INI=CALL W0JV
092 esav INI=CONF SI5351
093 esav INI=CONF 8MA CLK0
094 esav INI=CONF not used
095 esav INI=FREQ 144.150
096 esav INI=BATR
097 esav INI=MODS S0 360,0
098 esav INI=MODS S1 360,0
099 esav INI=REM- MODS S2 360,0
100 esav INI=REM- MODS S3 360,0
101 esav INI=REM- MODS S4 360,0
102 esav INI=REM- MODS S5 360,0
103 esav INI=MODS S6 600,0
104 esav INI=MODS S7 360,0
105 esav INI=MODS S8 360,0
106 esav INI=MODS S9 360,15
107 esav INI=STAT
108 esav TEST=FREQ 144.100
109 esav TEST=BEGN,SILENT
110 esav MAS=CWPM 35,-1,-1,-1,-1
111 esav MAS=STAT
112 esav REM- fox_ann_V2025.fox
113 esav ANN=TONE 1.0
114 esav ANN=CWPM 30,-1,-1,-1,-1
115 esav ANN=BEGN
116 esav ANN=BATR
117 esav ANN=TALK <CALL>
118 esav ANN=TALK <NAME>
119 esav ANN=WAIT 1.0
120 esav ANN=BATV V 7.2
121 esav ANN=BATV I
122 esav ANN=TIRP ASN
123 esav ANN=WAIT 0.3
124 esav ANN=TALK V_F144
125 esav ANN=TALK V_F225
126 esav ANN=TONE 1.0
127 esav ANN=CWPM 30,-1,-1,-1,-1
128 esav ANN=DONE
129 esav ANN=FREQ 144.225
130 esav ANN=STAT
131 esav ANN=RUN0 S6
132 esav ID=FOX2X_S0.fox Baseline
133 esav S0=BATR
134 esav S0=CONF FM
135 esav S0=TONE 1.0
136 esav S0=CWPM 30,-1,-1,-1,-1
137 esav S0=BEGN
138 esav S0=TALK <CALL>
139 esav S0=TALK <NAME>
140 esav S0=WAIT 0.5
141 esav S0=TONE 1.3
142 esav S0=CWPM 25,-1,-1,-1,-1
143 esav S0=WAIT 0.15
144 esav S0=BATC EV 7.2
145 esav S0=WAIT 0.5
146 esav S0=CODE IOWA CITY
147 esav S0=CODE AMATEUR RADIO
148 esav S0=CODE CLUB FOXHUNT
149 esav S0=CODE F W KENT PARK
150 esav S0=CODE MARCH 30 2025
151 esav S0=REM-
152 esav S0=REM-
153 esav S0=REM-
154 esav S0=BATR
155 esav S0=TONE 1.0
156 esav S0=CWPM 30,-1,-1,-1,-1
157 esav S0=DONE
158 esav ID=FOX2X_S1.fox CHiRP
159 esav S1=TONE 1.0
160 esav S1=CWPM 30,-1,-1,-1,-1
161 esav S1=BEGN
162 esav S1=TALK <CALL>
163 esav S1=TALK <NAME>
164 esav S1=CONF CW
165 esav S1=CHRP 1.0 6,0 0.30 55
166 esav S1=CONF FM
167 esav S1=TONE 1.0
168 esav S1=CWPM 30,-1,-1,-1,-1
169 esav S1=DONE SILENT
170 esav ID=FOX2X_S2.fox _2025_TREK.hex
171 esav S2=BEGN
172 esav S2=TALK <CALL>
173 esav S2=TALK <NAME>
174 esav S2=TALK HEY_LARRY
175 esav S2=TALK HEY_MOE
176 esav S2=TALK SORRY_MOE
177 esav S2=TALK CURLY_THINKS
178 esav S2=TALK BIG_IDEA
179 esav S2=TALK 3S_TOUPEE
180 esav S2=TALK 3S_WISE_GUYA
181 esav S2=TALK 3S_HEY_MOE1
182 esav S2=TALK 3S_HEY_MOE2
183 esav S2=TALK 3S_HEY_MOE3
184 esav S2=TALK 3S_VICTIM1
185 esav S2=TALK 3S_TAXIDERMIST
186 esav S2=TALK 3S_PAUSE
187 esav S2=DONE
188 esav ID=FOX2X_S3.fox _2025_TREK.hex
189 esav S3=BEGN
```

```
190 esav S3=TALK <CALL>
191 esav S3=TALK <NAME>
192 esav S3=TALK 2K1_H_9000
193 esav S3=TALK 2K1_GD_EVE
194 esav S3=TALK 2K1_CHESS2
195 esav S3=TALK 2K1_ENJOYA
196 esav S3=TALK 2K1_JUST_MOM
197 esav S3=TALK 2K1_MSG_4_U
198 esav S3=TALK 2K1_MSG_REP
199 esav S3=TALK 2K1_IGNIT
200 esav S3=TALK 2K1_DANGER
201 esav S3=TALK 2K1_FOOLPROOF
202 esav S3=TALK 2K1_HUMAN_ERR
203 esav S3=DONE
204 esav ID=FOX2X_S4.fox _2025_TREK.h24
205 esav S4=BEGN
206 esav S4=TALK <CALL>
207 esav S4=TALK <NAME>
208 esav S4=TALK TREK_YELLOW
209 esav S4=TALK TREK_AYE_SIR
210 esav S4=TALK TREK_ABSORPT
211 esav S4=TALK TREK_GREETIN
212 esav S4=TALK TREK_HAILING
213 esav S4=TALK TREK_MCCOY_
214 esav S4=TALK TREK_QUESTION
215 esav S4=TALK TREK_SQRE_NOST
216 esav S4=TALK TREK_SQRE_UNUS
217 esav S4=DONE
218 esav ID=FOX2X_S5.fox Wildlife
219 esav S5=TONE 1.0
220 esav S5=CWPM 30,-1,-1,-1,-1
221 esav S5=BEGN
222 esav S5=TALK <CALL>
223 esav S5=TALK <NAME>
224 esav S5=CONF CW
225 esav S5=WAIT 3
226 esav S5=CHRP 1.0 12 0 0.100 46
227 esav S5=WAIT 3
228 esav S5=CONF FM
229 esav S5=TONE 1.0
230 esav S5=CWPM 30,-1,-1,-1,-1
231 esav S5=DONE
232 esav ID=FOX2X_S6.fox CHiRP
233 esav S6=TONE 1.0
234 esav S6=CWPM 30,-1,-1,-1,-1
235 esav S6=BEGN
236 esav S6=TALK <CALL>
237 esav S6=TALK <NAME>
238 esav S6=CONF CW
239 esav S6=WAIT 3
240 esav S6=CHRP CHIRP_UP 12 0 0.05 23
241 esav S6=CHRP CHIRP_DN 12 0 0.05 23
242 esav S6=WAIT 3
243 esav S6=CONF FM
244 esav S6=TONE 1.0
245 esav S6=CWPM 30,-1,-1,-1,-1
246 esav S6=DONE
247 esav ID=FOX2X_S7.fox Field Day
248 esav S7=CONF FM
249 esav S7=TONE 1.0
250 esav S7=CWPM 30,-1,-1,-1,-1
251 esav S7=BEGN
252 esav S7=TALK <CALL>
253 esav S7=TALK <NAME>
254 esav S7=WAIT 1.0
255 esav S7=TALK FD_W0JV
256 esav S7=WAIT 1.0
257 esav S7=CWPM 30,-1,-1,-1,-1
258 esav S7=TONE 1.5
259 esav S7=CODE IOWA CITY
260 esav S7=CODE AMATEUR RADIO CLUB
261 esav S7=CODE FIELD DAY OPERATIONS
262 esav S7=CODE AT F W KENT PARK
263 esav S7=WAIT 1.0
264 esav S7=TALK FD_GAZELLE
265 esav S7=WAIT 1.0
266 esav S7=REM-
267 esav S7=REM-
268 esav S7=REM-
269 esav S7=TONE 1.0
270 esav S7=CWPM 30,-1,-1,-1,-1
271 esav S7=DONE
272 esav ID=FOX2X_STOOGE.fox S8 Shemp
273 esav S8=TONE 1.0
274 esav S8=CWPM 30,-1,-1,-1,-1
275 esav S8=CONF FM
276 esav S8=BEGN
277 esav S8=TALK <CALL>
278 esav S8=TALK <NAME>
279 esav S8=CODE SING SING
```

```
280 esav S8=CONF CW
281 esav S8=CODE E
282 esav S8=WAIT 120/40
283 esav S8=TALK TS1_LA
284 esav S8=WAIT 120/50
285 esav S8=TALK TS2_NY
286 esav S8=WAIT 120/60
287 esav S8=TALK TS3_SING
288 esav S8=WAIT 120/70
289 esav S8=TALK TS4_BOSTON
290 esav S8=WAIT 120/80
291 esav S8=CONF FM
292 esav S8=TONE 1.2
293 esav S8=CODE BEEN LISTENIN TO
294 esav S8=CODE THE THREE STOOGES
295 esav S8=TONE 1.0
296 esav S8=CWPM 30,-1,-1,-1,-1
297 esav S8=DONE
298 esav ID=FOX2X_STOOGE.fox S9 Moe
299 esav S9=TONE 1.0
300 esav S9=CWPM 30,-1,-1,-1,-1
301 esav S9=CONF FM
302 esav S9=BEGN
303 esav S9=TALK <CALL>
304 esav S9=TALK <NAME>
305 esav S9=CODE SING SING
306 esav S9=CONF CW
307 esav S9=CODE E
308 esav S9=WAIT 120/44
309 esav S9=TALK TS1R_LA
310 esav S9=WAIT 120/54
311 esav S9=TALK TS2R_NY
312 esav S9=WAIT 120/64
313 esav S9=TALK TS3R_SING
314 esav S9=WAIT 120/74
315 esav S9=TALK TS3R_SING
316 esav S9=WAIT 120/100
317 esav S9=CONF FM
318 esav S9=TONE 1.6
319 esav S9=CODE BEEN LISTENIN TO
320 esav S9=CODE THE THREE STOOGES
321 esav S9=TONE 1.0
322 esav S9=CWPM 30,-1,-1,-1,-1
323 esav S9=DONE
324 esav ID=FT,si5351_frq_cmds1_8.fox
325 esav 144.FOFF -8.000
326 esav 144.100=139D,206C0,F4240
327 esav 144.105=139D,4F4C0,F4240
328 esav 144.110=139D,7E2BF,F4240
329 esav 144.115=139D,AD0C0,F4240
330 esav 144.120=139D,DBEBF,F4240
331 esav 144.125=139E,16A80,F4240
332 esav 144.130=139E,45880,F4240
333 esav 144.135=139E,7467F,F4240
334 esav 144.140=139E,A347F,F4240
335 esav 144.145=139E,D2280,F4240
336 esav 144.150=139F,0CE3F,F4240
337 esav 144.155=139F,3BC40,F4240
338 esav 144.160=139F,6AA40,F4240
339 esav 144.165=139F,9983F,F4240
340 esav 144.170=139F,C863F,F4240
341 esav 144.175=13A0,03200,F4240
342 esav 144.180=13A0,31FFF,F4240
343 esav 144.185=13A0,60E00,F4240
344 esav 144.190=13A0,8FC00,F4240
345 esav 144.195=13A0,BE9FF,F4240
346 esav 144.200=13A0,ED7FF,F4240
347 esav 144.205=13A1,283C0,F4240
348 esav 144.210=13A1,571BF,F4240
349 esav 144.215=13A1,85FC0,F4240
350 esav 144.220=13A1,B4DC0,F4240
351 esav 144.225=13A1,E3BBF,F4240
352 esav 144.230=13A2,1E77F,F4240
353 esav 144.235=13A2,4D580,F4240
354 esav 144.240=13A2,7C37F,F4240
355 esav 144.245=13A2,AB180,F4240
356 esav 144.250=13A2,D9F80,F4240
357 esav 144.255=13A3,14B3F,F4240
358 esav 144.260=13A3,4393F,F4240
359 esav 144.265=13A3,72740,F4240
360 esav 144.270=13A3,A153F,F4240
361 esav 144.275=13A3,D0340,F4240
362 esav 144.280=13A4,0AEFF,F4240
363 esav 144.285=13A4,39CFF,F4240
364 esav 144.290=13A4,68AFF,F4240
365 esav 144.295=13A4,978FF,F4240
366 esav 144.300=13A4,C66FF,F4240
367 esav 144.305=13A5,012C0,F4240
368 esav 144.310=13A5,300BF,F4240
369 esav 144.315=13A5,5EEBF,F4240
370 esav 144.320=13A5,8DCBF,F4240
371 esav 144.325=13A5,BCABF,F4240
372 esav 144.330=13A5,EB8BF,F4240
373 esav 144.335=13A6,26480,F4240
374 esav 144.340=13A6,5527F,F4240
375 esav 144.345=13A6,8407F,F4240
376 esav 144.350=13A6,B2E7F,F4240
377 esav ID=FT,si5351_frq_cmds2_8.fox
378 esav 144.375=13A7,A923F,F4240
379 esav 144.400=13A8,9F5FF,F4240
```

```
380 esav 144.425=13A9,959C0,F4240
381 esav 144.450=13AA,8BD7F,F4240
382 esav 144.475=13AB,8213F,F4240
383 esav 144.500=13AC,784FF,F4240
384 esav 144.525=13AD,6E8C0,F4240
385 esav 144.550=13AE,64C80,F4240
386 esav 144.575=13AF,5B040,F4240
387 esav 144.600=13B0,51400,F4240
388 esav 144.625=13B1,477C0,F4240
389 esav 144.650=13B2,3DB80,F4240
390 esav 144.675=13B3,33F3F,F4240
391 esav 144.700=13B4,2A300,F4240
392 esav 144.725=13B5,206C0,F4240
393 esav 144.750=13B6,16A80,F4240
394 esav 144.775=13B7,0CE3F,F4240
395 esav 144.800=13B8,03200,F4240
396 esav 144.825=13B8,ED7FF,F4240
397 esav 144.850=13B9,E3BBF,F4240
398 esav 144.875=13BA,D9F80,F4240
399 esav 144.900=13BB,D0340,F4240
400 esav 144.925=13BC,C66FF,F4240
401 esav 144.950=13BD,BCABF,F4240
402 esav 144.975=13BE,B2E7F,F4240
403 esav ID=FT,si5351_frq_cmds99_4.fox
404 esav 87.500=0F80,32000,F4240,200
405 esav 87.700=0F8A,6C980,F4240,200
406 esav 87.900=0F94,A72FF,F4240,200
407 esav 88.100=0F9E,E1C7F,F4240,200
408 esav 88.300=0FA9,283BF,F4240,200
409 esav 88.500=0FB3,62D40,F4240,200
410 esav 88.700=0FBD,9D6C0,F4240,200
411 esav 88.900=0FC7,D803F,F4240,200
412 esav 89.100=0FD2,1E77F,F4240,200
413 esav 89.300=0FDC,59100,F4240,200
414 esav 89.500=0FE6,93A80,F4240,200
415 esav 89.700=0FF0,CE3FF,F4240,200
416 esav 89.900=0FFB,14B3F,F4240,200
417 esav 90.100=1005,4F4C0,F4240,200
418 esav 90.300=100F,89E40,F4240,200
419 esav 90.500=1019,C47BF,F4240,200
420 esav 90.700=1024,0AEFF,F4240,200
421 esav 90.900=102E,45880,F4240,200
422 esav 91.100=1038,80200,F4240,200
423 esav 91.300=1042,BAB7F,F4240,200
424 esav 91.500=104D,012BF,F4240,200
425 esav 91.700=1057,3BC40,F4240,200
426 esav 91.900=1061,765C0,F4240,200
427 esav 92.100=106B,B0F3F,F4240,200
428 esav 92.300=1075,EB8BF,F4240,200
429 esav 92.500=1080,32000,F4240,200
430 esav 92.700=108A,6C980,F4240,200
431 esav 92.900=1094,A72FF,F4240,200
432 esav ID=FT,si5351_frq_cmds80_1.fox
433 esav 3.500=0D0D,DBEC0,F4240,5400
434 esav 3.550=0D17,77240,F4240,5300
435 esav 3.600=0D1F,C223F,F4240,5200
436 esav 3.650=0D26,C8C80,F4240,5100
437 esav 3.700=0D2C,8B0FF,F4240,5000
438 esav 3.750=0D01,05DBF,F4240,4E00
439 esav 3.800=0D03,8B73F,F4240,4D00
440 esav 3.850=0D04,CCB00,F4240,4C00
441 esav 3.900=0D04,C9900,F4240,4B00
442 esav 3.950=0D03,8213F,F4240,4A00
```

A few notes about the latest (2025-JUN-06 V4.07) updates to the commands loaded into the fox transmitters.

Line 324 we see the intermediate file that was generated with the frequency table for the bottom end of the 2M band.

Line 377 we add 25KHz steps to fill out the frequencies needed for a formal hunt. We don't currently pay a penalty for haveing them present as the FRAM in these transmitters are all 256KB or larger (1024 commands).

Line 403 we add the bottom of the FM broadcast band. This requires some reworking of the low pass filter on the main board, so it's not an easy switch from 2M to 3M (100MHz). We also need to reduce power when operating in the FM band to remain within radiatiated emission limits imposed by the FCC.

You will also notice that we pick up a fourth numeric parameter for setting up the SI5351. We change the divisor on the second MultiSynth to a **2** to account for the lower frequency.

Line 432 was added for 80M band operation with the 4.09 revision. As mentioned elsewhere, operating in this band requies the use of a band-specific RF amplifier daughterboard (this board includes an LPF appropriate for the band) Earlier software will not allow this selection to be effected (it will be rejected!).

Operating down in the 80M band also makes much greated use of the second multisynth stage. The divisor becomes much larger here!

Chapter 20

Sample Output

Sample output.

20.1 Sample HELP

Example Help listing.

Listing 20.1: FOX_ICARC_help.txt

```
09:51:11: RDY00,01* (Sp=0xBFD8)+1927 09:51:12.070                                                                  1
09:51:12: sts01,00* TEST HELP **  TEST HELP **  TEST HELP **                                                       2
09:51:12: sts01,00* Idx MNE  Class      ————————Arguments———————— Command Function                                3
09:51:12: sts01,01*   1 HELP SYS                                  Help Menu and Items                            4
09:51:12: sts01,02*   2 HELP SYS        <string>                  matching help items                            5
09:51:12: sts01,03*   3 ONCE SYS        <name>                    Test run the named seqwuence                   6
09:51:12: sts01,04*   4 REM- SYS                                  Remark, (side-effect: stops schedules)         7
09:51:12: sts01,05*   5 RUN0 SYS                                  RUN ALL Schedules                              8
09:51:12: sts01,06*   6 RUN0 SYS        <name>                    RUN Specific Schedule                          9
09:51:12: sts01,07*   7 STAR SYS        <time hh:mm:ss>           Start running schedules at specified time      10
09:51:12: sts01,08*   8 IDLE SYS                                  STOP ALL Schedules                             11
09:51:12: sts01,09*   9 STAT SYS        <flag>                    System Status, (I)ident scan                   12
09:51:12: sts01,10*  10 CONF SYS        <keywords>                Hardware Configuration                         13
09:51:12: sts01,11*  11 TOYC SYS        <res> (250 2K 4K NONE)    Hi chg rte DS1672 bat                          14
09:51:12: sts01,12*  12 TIME SYS        <time value>              Set Time (set DS1672)                          15
09:51:12: sts01,13*  13 D525 SYS        <sub-command>             ICS525 debug routines                          16
09:51:12: sts01,14*  14 TIME SETUP                                 Time from DS1672 to System (NO Argument!)      17
09:51:12: sts01,15*  15 EPOC SETUP      <hours>                   Epoch offset (i.e. time zone)                  18
09:51:12: sts01,16*  16 CALL SETUP      <call>                    FCC Assigned Callsign                          19
09:51:12: sts01,17*  17 NAME SETUP      <nick>                    Local Nickname                                 20
09:51:12: sts01,18*  18 NICK SETUP      <nick>                    alias for "NAME", but don't use it!            21
09:51:12: sts01,19*  19 TONE PGM        <freq>)                   Audio Tone (in KHz)                            22
09:51:12: sts01,20*  20 CWPM PGM        <wpm gap1 gap2 gap3>      CW Chipping Rate                               23
09:51:12: sts01,21*  21 FREQ PGM        <freq>                    Frequency (in MHz)                             24
09:51:12: sts01,22*  22 FOFF PGM        <offset>                  Frequency Offset (in KHz)                      25
09:51:12: sts01,23*  23 5351 PGM        <key>,<value>,<value>,... SI5351 setup group                             26
09:51:12: sts01,24*  24 BEGN PGM                                   Key TX and Send Callsign (CW)                  27
09:51:12: sts01,25*  25 CODE PGM        <message>                 Send Message (CW) up to 22 char                28
09:51:12: sts01,26*  26 TALK PGM        <file-name>               Play Voiced Message (EDMP TALK)                29
09:51:12: sts01,27*  27 WAIT PGM        <secon.ds>                Wait (simple delay)                            30
09:51:12: sts01,28*  28 CHRP PGM        <tone> <per> <dur> <cnt>  Send carrier chirp                             31
09:51:12: sts01,29*  29 DONE PGM                                   Send Callsign (CW), SK (CW), and unkey TX      32
09:51:12: sts01,30*  30 BATC PGM        <mod>,<key>,<setpoint>    Transmit Code Battery Report                   33
09:51:12: sts01,31*  31 BATV PGM        <mod>,<key>               Transmit Vocal Battery Report                  34
09:51:12:                                                          mod: "E" encode (not CW) for BATC              35
09:51:12:                                                          mod: "B" battery reading taken before BEGN     36
09:51:12:                                                          mod: "A" battery reading taken after BEGN      37
09:51:12:                                                          key: "V" voltage, "I" current, "R" 5V rail     38
09:51:12: sts01,32*  32 MODS SCHED      <Sname period offset>     Modulus Schedule Set                           39
09:51:12: sts01,33*  33 MODC SCHED      <Sname=>                  Modulus Schedule Clear                         40
09:51:12: sts01,34*  34 TALK DIRECTORY esav TALK=name,Strt,Len,rate      (in FRAM as the TALK= file)               41
09:51:12:                                                          Waveform Directory Entry                       42
09:51:12:                                                          rate keys: 4K 5K 8K 10K 16K                    43
09:51:12: sts01,35*  35 freq DIRECTORY esav 144.150=13BF,70E40,F4240,100 (in FRAM as frequency record)            44
09:51:13:                                                          Register Params are Synthesizer dependant      45
09:51:13: sts01,36*  36 ESAV FRAM       NAM=<text>                Save named record in next free location        46
09:51:13: sts01,37*  37 EDMP FRAM       "match string"            Dump active records                            47
09:51:13: sts01,38*  38 EDID FRAM                                  Flash JEDEC-ID table dump (PROG & WAVE)        48
09:51:13: sts01,39*  39 ERAS FRAM       <number> or "DEV"         Rewrite <record> to REM- (DEV, QTR, HALF)      49
09:51:13: sts01,40*  40 EZER FRAM       <number>                  Erase <record> to ZERO                         50
```

Listing 20.2: FOX_ICARC_help.txt-1

```
09:51:13: sts01,41*  41 ETAB FRAM                                  Dump JEDEC-ID device table                                    51
09:51:13: sts01,42*  42 HERA FLASH      ALL                       Hex erase (entire WAVE device)                                52
09:51:13: sts01,43*  43 HDMP FLASH      <len-32B-lines <hex-start <*>>> Hex dump (WAVE device)                                  53
09:51:13: sts01,44*  44 HEND FLASH                                 Find end of waveform data                                     54
09:51:13: sts01,45*  45 H115 FLASH                                 Fast terminal bit rate                                        55
09:51:13: sts01,46*  46 :hex FLASH-HEX :llaaaattddddddddcc         Intel HEX loader (WAVE device)                                56
09:51:13: sts01,47*  47 HALT TEST                                  Halt Processor                                                57
09:51:13: sts01,48*  48 STOP TEST                                  Stop Processor                                                58
09:51:13: sts01,49*  49 REST TEST                                  Reset System                                                  59
09:51:13: sts01,50*  50 TEST TEST                                  Hardware Test Subsystem                                       60
09:51:13: STS01,51* Handler_HELP (cmd_help.c*)  1.00 Sec                                                                         61
09:51:13: RDY00,00* (Sp=0xBF94)+1859 09:51:14.190                                                                                62
```

20.2 Sample STAT

Example Stat Listing.

Listing 20.3: FOX_ICARC_stat.txt

```
12:04:45: sts09,00* <<<---  STAT  ********  STAT  --->>>                                                                        1
12:04:45: sts09,01* KC0JFQ  FOX Transmitter  V3.88                                                                               2
12:04:45: sts09,02* Software Bld:    Feb  7 2025  13:33:08                                                                       3
12:04:45: sts09,03* System Time:     13:04:46.160 (65086)                                                                        4
12:04:45: sts09,04* Epoch Offset:    19:00:00     (68400)                                                                        5
12:04:45: sts09,05* TOY Clock:       000CDBBE 00 04; Osc ON, Charge Disabled                                                     6
12:04:45: sts09,06* Sys Upd Flg:     Si5351_INIT                                                                                 7
12:04:45: sts09,07* Conf Jumpers:    NOT_Master NOT_Test                                                                         8
12:04:45: sts09,08* Flash Prog  U3: 04.25.03 MB85RS256TY    FLASH_FRAM       Fujitsu   256K-bits   1024-records CMD3              9
12:04:45: sts09,09* Flash WAVE U12: 9D.7E.FF    25LD040    FLASH_PAGE         ISSI  4096K-bits     62-seconds CMD4              10
12:04:45: sts09,10* Flash HEX Dev:  WAVE/FLASH/U12                                                                               11
12:04:45: sts09,11* Battery, Idle:   8.805V(0x0387)[9.751e-03]  23mA(0x002F)                                                     12
12:04:45: sts09,12* Battery, TX:     8.620V(0x0374)[9.751e-03]  106mA(0x00D9)                                                    13
12:04:45: sts09,13* Analog Others:   Reg+5V: 5.090V(0x020A)[9.751e-03]  Switch: 0.000V  CdS-Cell: 0.000V                         14
12:04:45: sts09,14* UART buffer:     175 (NET:0, USB:96)                                                                         15
12:04:45: sts09,15* <<<--- Scheduling  PARAMETERS --->>>                                                                         16
12:04:45: sts09,16* MOD Schedule     00 Idle     S0= 360    0                                                                    17
12:04:45: sts09,17* MOD Schedule     01 Idle     S1= 360    0                                                                    18
12:04:45: sts09,18* MOD Schedule     02 Idle     S6= 600    0                                                                    19
12:04:45: sts09,19* MOD Schedule     03 Idle     S7= 360    0                                                                    20
12:04:45: sts09,20* MOD Schedule     04 Idle     S8= 360    0                                                                    21
12:04:45: sts09,21* MOD Schedule     05 Idle     S9= 360   15                                                                    22
12:04:45: sts09,22* Run Start:       13:05:59 Waiting (start:47159 now:47086)                                                    23
12:04:45: sts09,23* <<<--- TRANSMITTER PARAMETERS --->>>                                                                         24
12:04:45: sts09,24* Callsign:        W0JV                                                                                        25
12:04:45: sts09,25* Nickname:        FOX21                                                                                       26
12:04:45: sts09,26* zNEO Port Bits: OUT:E0,01,05,20,00,00,00,00 IN:F0,01,75,14,00,80,00,04                                       27
12:04:45: sts09,27* Radio Config:    0x1F0C43 SI5351 State-T0 TX_ENA TONE PWMH0 5MON SWIT PHOTO IMON VMON                        28
12:04:45: sts09,28* Si5351 Config:   0x00B9 CLK0 8MA 8PF 0x00B9                                                                  29
12:04:45: sts09,29* Si5351 Divisor: 0x13A1,0xE3BBF,0xF4240 0x0100,0x00,0x01                                                      30
12:04:45: sts09,30* Frequency:       144.225 (Xtal: 20.000 MHz) Offset=-8.000KHz                                                 31
12:04:45: sts09,31* CW config:       30 WPM 1,3,7,14 [0x186A] 1.000KHz                                                           32
12:04:45: sts09,32* State Delays:    T0:10 T1:50 T2:150 T4:50 T5:10                                                              33
12:04:45: STS09,33* Handler_STAT I (cmd_stat.c*)  0.42 Sec                                                                       34
```

20.3 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
```

Chapter 21

Power Worksheets

Power Measurements from working units.

21.1 FOX6

This station is using the 102-73161-25 board. This power amp is a dual logic gate implementation.

UNIT NAME	Artwork Revision	Power Amp Revision	Output Power
FOX6	102-73161-25	102-73161-24	40mW
Idle Voltage	**Idle Current**	**Active Voltage**	**Active Current**
9.50V	15mA	9.0V	58mA

Table 21.1: FOX6/102-73161-25

Additional information on this amplifier is found on page 77.

21.2 FOX22

This station is equipped with the DRA818 tranceiver module. (it may be operated as a wildlife tracker, but it's a power pig).

UNIT NAME	Artwork Revision	Power Amp Revision	Output Power
FOX22	102-73181-10	102-73181-36	100mW
Idle Voltage	**Idle Current**	**Active Voltage**	**Active Current**
9.50V	21mA	8.0V	250mA

Table 21.2: FOX22/102-73181-10

More information on the power amp may be found on page 94.

21.3 FOX27

This station was used to test with several RF amplifiers. (will **not** operate as a wildlife tracker).

UNIT NAME	Artwork Revision
FOX27	102-73181-10

Table 21.3: FOX27/102-73181-10

PA Revision	Output Power	Idle V	Idle I	Active V	Active I
102-73161-28	30mW	9.50V	20mA	9.1V	65mA
102-73161-28	70mW	9.50V	20mA	9.1V	80mA
102-73161-28	100mW	9.50V	25mA	9.1V	90mA
102-73161-29	35mW	9.50V	20mA	9.1V	60mA
102-73181-28	50mW	9.50V	20mA	9.1V	90mA
102-73181-28	50mW	9.50V	16mA	9.1V	85mA
102-73181-28	110mW	9.50V	20mA	9.1V	100mA

Table 21.4: FOX27/102-73181-AMPS

Additional information on the 102-73161-28 amplifier is found on page 80.
Additional information on the 102-73161-29 amplifier is found on page 78.
Additional information on the 102-73181-28 amplifier is found on page 82.

21.4 FOX29

This station is equipped with the basic MMIC amplifier (will **not** operate as a wildlife tracker).

UNIT NAME	Artwork Revision	Power Amp Revision	Output Power
FOX29	102-73181-10	102-73161-28	50mW
Idle Voltage	**Idle Current**	**Active Voltage**	**Active Current**
9.50V	16mA	9.1V	85mA

Table 21.5: FOX29/102-73181-10

Additional information on the amplifier is found on page 80.

21.5 FOX32

This station is equipped to operate in both the *normal* mode (i.e. voice and CW) as well as wildlife tracker mode.

UNIT NAME	Artwork Revision	Power Amp Revision	Output Power
FOX32	102-73181-10	102-73181-28	40mW
Idle Voltage	**Idle Current**	**Active Voltage**	**Active Current**
9.50V	16mA	9.1V	85mA

Table 21.6: FOX32/102-73181-10

Additional information on thie amplifier is found on page 82.

Chapter 22

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.

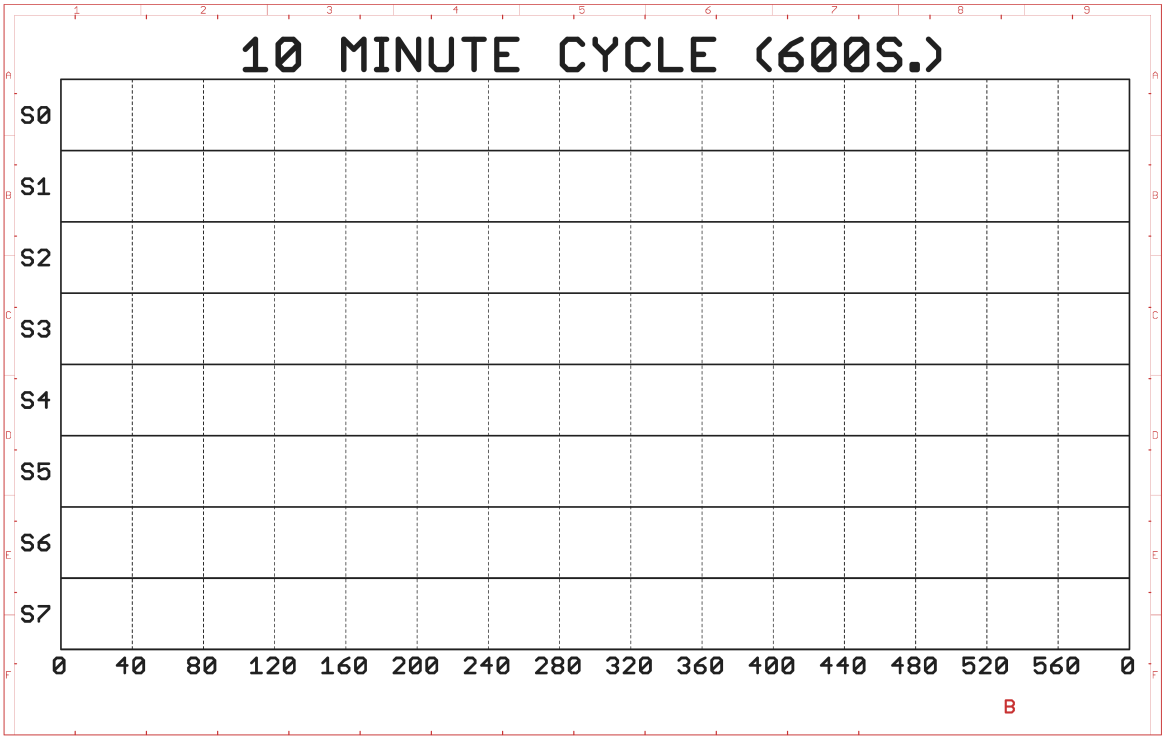


Figure 22.1: Worksheet, 10 minute cycle

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.

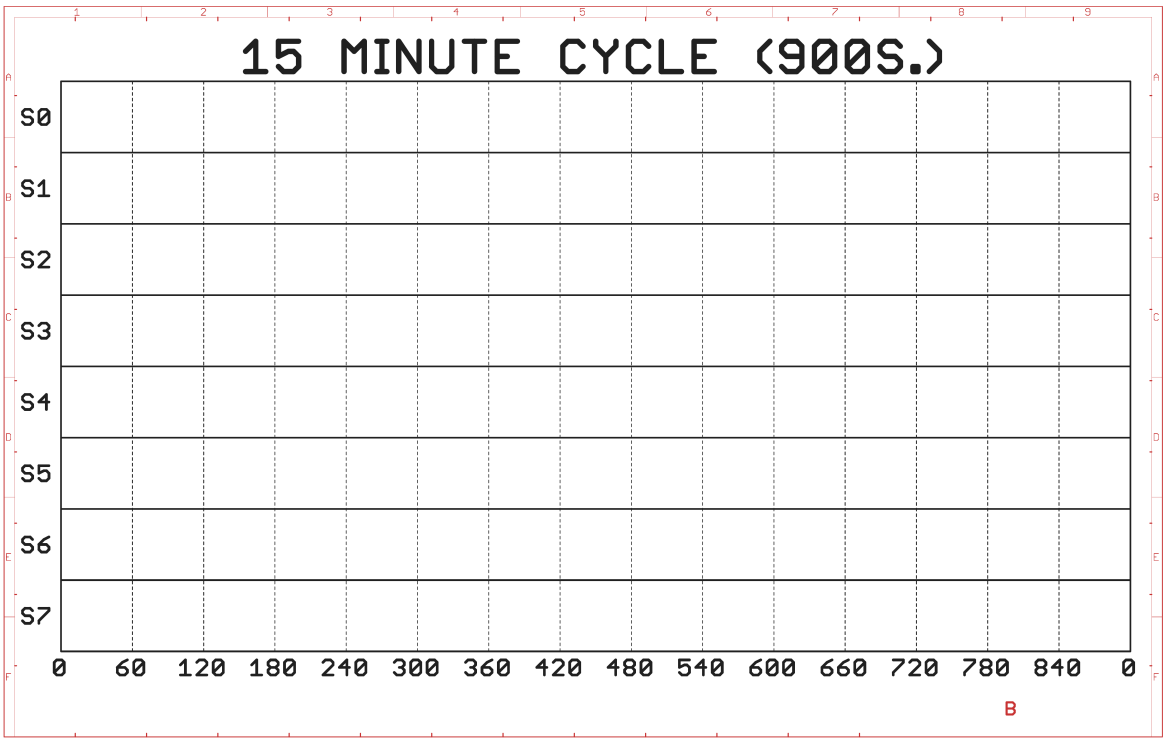


Figure 22.2: Worksheet, 15 minute cycle

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.

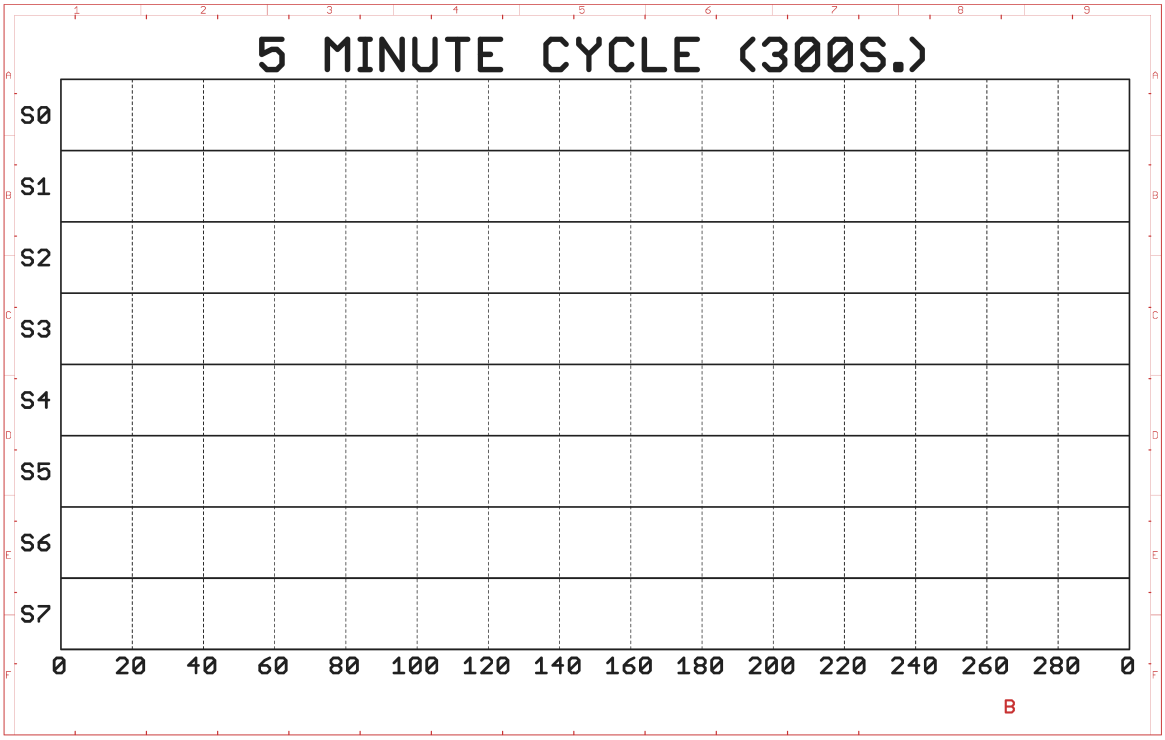


Figure 22.3: Worksheet, 5 minute cycle

This is an example schedule as it is worked out on the worksheet.

Here we replay the old *Three Stooges* "You Are Now in Sing-Sing" gag on two stations. Four other stations are members of this hunt group and they must all play together.

This is possible using two assumptions:

First

The clocks are all synchronized the night prior to the hunt and that they track reasonably well (within a second or two per day).

Second

The software in the fox transmitter is at or above V3.76. This revision improved the way the **TIME** gets time from the DS1672 TOY clock. The end result being the fox transmitter times run within 10s of milliseconds of each-other.

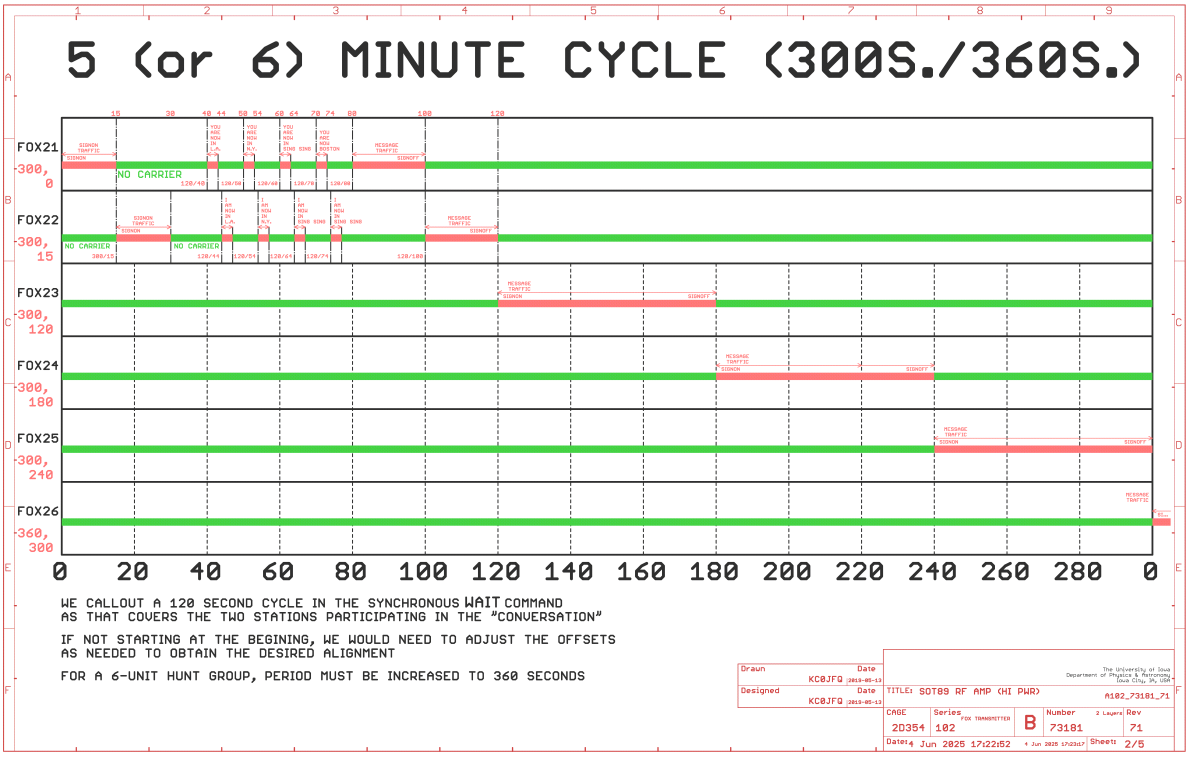


Figure 22.4: Conversation, 5 or 6 minute cycle

This timing chart shows activity where **FOX21** is using the S8= sequence and **FOX22** is using the S9= sequence. The remaining transmitters in the group **FOX23**..**FOX26** are all using the S0= sequence. Timing for S0, S8, and S9 are all a six minute period and the transmitters are sequentially staggered every 60 seconds. The S8 and S9 sequence can be found in section on page 346.

FOX23..**FOX26** all run on a nor mal schedule and simply send the CW message out after station identification. Following the CW message we send a final station identification and the go silent until next time around.

FOX21 and **FOX22** attempt to reenact a sceen from an old *Three Stooges* short where Shemp hypnotises Moe. As Shemp take Moe around the country in his hypnotised State. When Shemp calls out: "You are now in Sing-Sing", Moe picks up a straight-backed char and replies" "I am now in Sing-Sing". Shemp moves on to Boston: "You are now in Boston". Moe, however, is stuck (behind bars formes by the back of the chair): "I am now in Sing-Sing".

FOX21 plays the the part of Shemp, calling out the part of the country Moe now occupies.

FOX22 plays the the part of Moe, providing confirmation that the hypnotic trance has been effective.

FOX21 and FOX22 are interleaved for this call&reply interaction. That is they are operating in close synchronization requiring an accurate time in the TOY clock. This seems to work well if the TOY clock has been updated the day prior to the hunt.

FOX23..FOX26 operate normally using the S0 sequence.

We can zoom in on the **FOX21 and FOX22** activity such that you get an idea of the timing needed for this to work.

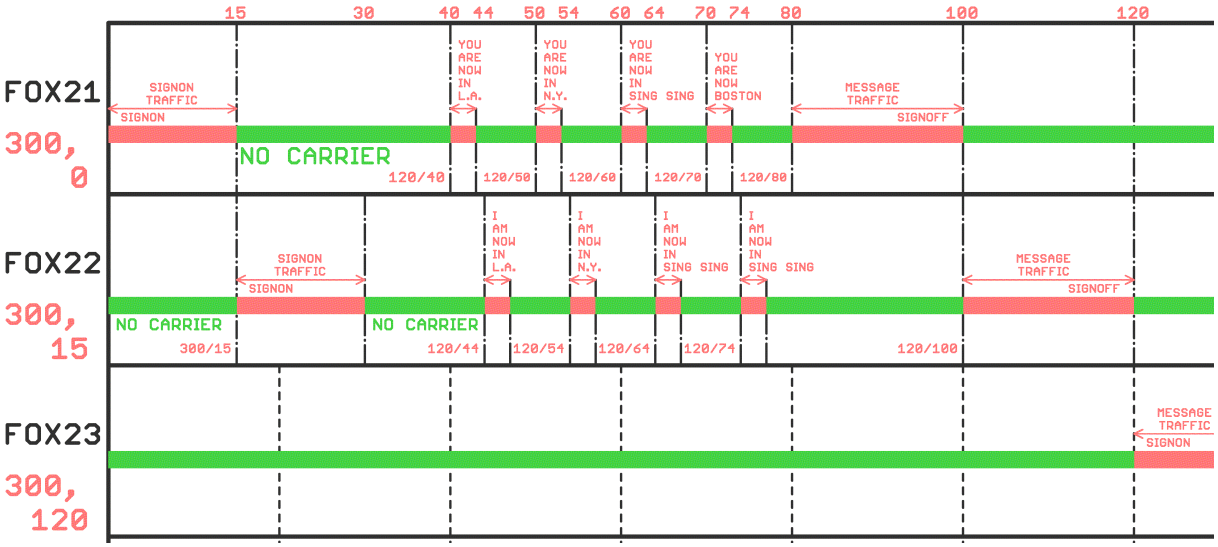


Figure 22.5: Conversation, FOX21/FOX22

FOX21 and **FOX22** get their signon traffic out of the way and then settle into the interleaved conversation. You can refer to the **SX_STOOGE.fox** sequence in section on page 346.

After the signon message we specify a synchronous wait with: **WAIT 120/40** on **FOX21** and **WAIT 120/44** on **FOX22**. The period of 120 was selected as this is the time allocated to **FOX21 and FOX22**. Since these are running on a 360 second cycle, the 120 second merges seamlessly.

The signoff activity is similarly scheduled to get the **FOX21** signoff to occur first followed by the **FOX22** signoff.

Example of an interleaved group (the S6 schedule)

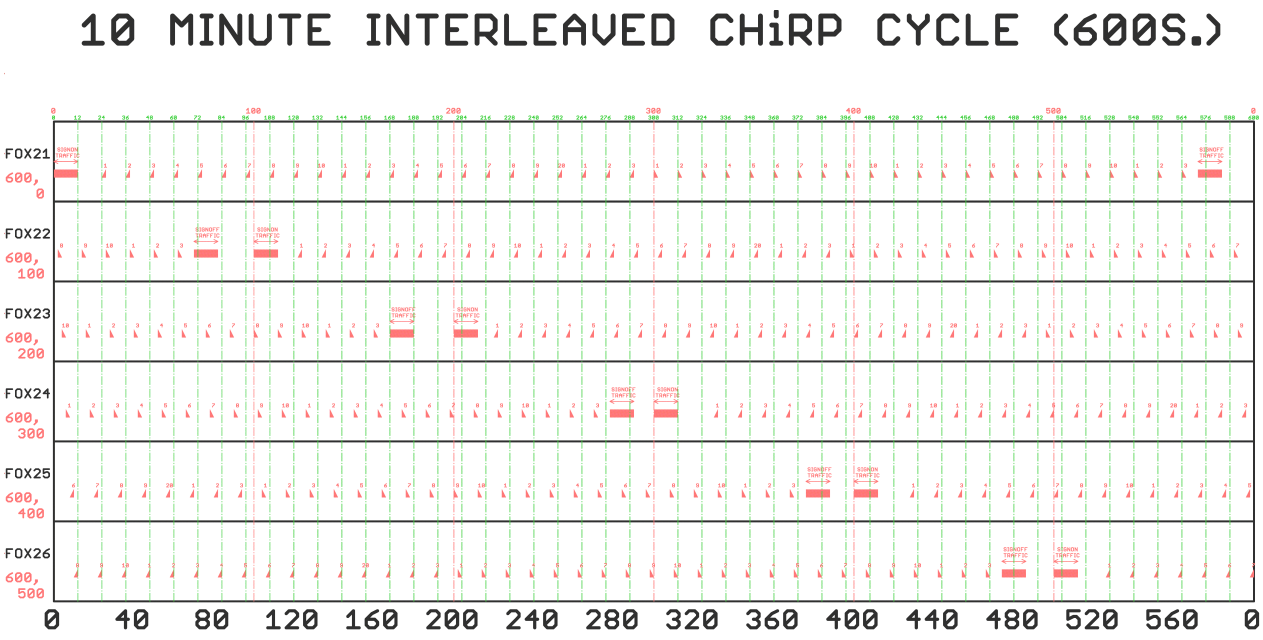


Figure 22.6: CHiRP, FOX21..FOX26

This is a difficult fox hunting mode used to illustrate the fine-grained timing capability of the fox transmitter system. This mode depends on the TOY clock providing reasonable timing accuracy which is not difficult to achieve. The sequence shown, S6, can be seen in section 19.11 on page 341.

We begin by making the fox transmitter identify every 10 minutes, as indicated by the signon/signoff message in figure 22.6. This is to satisfy FCC part-95 station identification requirements.

The **BEGN** command sends our callsign in code which is followed by the **TALK <CALL>** and **TALK <NAME>** commands to verbally identify.

The **DONE** command that occurs after chirping for just over 9 minutes. This is to satisfy FCC part-95 to identify at the end of a transmission. We leave the off verbalization to minimize the on-air time we aren't chirping.

The main body of the *message* consists of an audio chirp where the audio tone changes between a few hundred cycles/second and a few thousand cycles/second. The start and stop frequency having been chosen to keep the signal bandwidth within what is allowed and what the transmitter is capable of. For the first 4+ minutes the audio tone climbs in frequency (an up-sweep) while in the last 4+ minutes the audio tone falls in frequency (a dowen-sweep).

The timing chart, then, shows the transmitting schedule we are attempting to follow. The chart shows a full 10-minute cycle, so you can think of duplicating the 10 minute chart to the right repeatedly.

The **FOX21** line illustrates the full cycle each transmitter goes through. We start, of course, with the signon **BEGN**, **TALK <CALL>**, and **TALK <NAME>**. This signon activity takes about 12 seconds to execute. The **FOX21** unit uses the following chirp commands: **CHRP CHIRP_UP 12 0 0.05 23** and **CHRP CHIRP_DN 12 0 0.05 23** to send out 46 chirps (taking 9.2 minutes). After th 46 chirps are sent we send our signoff with only the station callsign (which takes less than 10 econds).

The **FOX22** line is the same but shifted, in tim, by 100 seconds. You see, from the chart, that the **FOX22** signon message occurs, as expected, 100 seconds into the (synchronous) 600 second cycle.

You should also notice that the chirps are shifted to the right by 2 seconds. This comes about from the following chirp commands: **CHRP CHIRP_UP 12 2 0.05 23** and **CHRP CHIRP_DN 12 2 0.05 23.**

The argument after the period (i.e. the 12) is changed to 2 to shift the chirp off from on top of the chirp from **FOX21**.

The schedule for **FOX23** has a scheduling offset of 200 seconds, again visible in the chart. The chirp commands loaded into **FOX23** are further offset as follows: **CHRP CHIRP_UP 12 4 0.05 23** and **CHRP CHIRP_DN 12 4 0.05 23.**

The argument after the period (i.e. the 12) is changed here to 4 shifting the chirp out from under **FOX21** and **FOX22**.

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.

ICARC Fox Hunt

Transmitter Time Synchronization Checklist (V3.13)

Sun Jun 8 20:38:45 2025 fox_label_A_chk.ps

Event Callsign: **W0JV** Start: End:

Name	Chk Freq Oper Freq	Drwg Nmbr Daughterboard	Power S/W Ver	Batt Voltage Run Schedule	Batt Current	Event Validation Code Event ID Code	FRAM FLASH
FOX21 Unit: 1	144.150MHz 144.225MHz	102_73181_10 102_73181_28	70mW V4.08	8.00V Idle Tx **7.60V** S6	24mA Idle Tx 122mA	144.225 / **5ISOYQR9** Off:1 ID: **093872**	256K 4096K
FOX22 Unit: 2	144.150MHz 144.225MHz	102_73181_10 102_73181_28	80mW V4.08	8.90V Idle Tx **8.60V** S6	38mA Idle Tx 123mA	144.225 / **BBTEST4Q** Off:20 ID: **013726**	256K 8192K
FOX23 Unit: 3	144.150MHz 144.225MHz	102_73181_10 102_73181_28	95mW V4.08	7.90V Idle Tx **7.20V** S6	54mA Idle Tx 157mA	144.225 / **CJ0712P9** Off:13 ID: **827718**	256K 8192K
FOX24 Unit: 4	144.150MHz 144.225MHz	102_73181_10 102_73181_28	60mW V4.08	9.20V Idle Tx **9.00V** S6	32mA Idle Tx 110mA	144.225 / **HJ3ZVX5N** Off:13 ID: **715030**	256K 4096K
FOX25 Unit: 5	144.150MHz 144.225MHz	102_73181_10 102_73181_28	50mW V4.08	8.80V Idle Tx **8.60V** S6	37mA Idle Tx 123mA	144.225 / **DBTH6IVN** Off:22 ID: **665050**	256K 4096K
FOX26 Unit: 6	144.150MHz 144.225MHz	102_73181_10 102_73181_28	80mW V4.08	8.20V Idle Tx **7.80V** S6	32mA Idle Tx 123mA	144.225 / **62AB5ZLI** Off:23 ID: **777214**	256K 8192K
FOX27 Unit: 7	144.150MHz 144.325MHz	102_73181_10 102_73181_28	115mW V4.08	8.60V Idle Tx **8.30V** S0	29mA Idle Tx 121mA	144.325 / **WP8WDLF6** Off:26 ID: **128769**	512K 65536K
FOX28 Unit: 8	144.150MHz 144.325MHz	102_73181_10 102_73181_28	100mW V4.08	8.00V Idle Tx **7.50V** S0	32mA Idle Tx 118mA	144.325 / **4B0OD4VE** Off:16 ID: **099374**	512K 65536K
FOX29 Unit: 9	144.150MHz 144.325MHz	102_73181_10 102_73181_28	100mW V4.08	6.70V Idle Tx **5.40V** S0	46mA Idle Tx 171mA	144.325 / **C1GH0Y7M** Off:-8639 ID: **263841**	512K 262144K
FOX30 Unit: 10	144.150MHz 144.325MHz	102_73181_10 102_73181_28	100mW V4.08	7.80V Idle Tx **7.20V** S0	47mA Idle Tx 151mA	144.325 / **J14K5PDG** Off:16 ID: **724191**	512K 262144K
FOX31 Unit: 11	144.150MHz 144.325MHz	102_73181_10 102_73181_28	50mW V4.08	7.60V Idle Tx **7.10V** S0	39mA Idle Tx 136mA	144.325 / **ME33HPFU** Off:1 ID: **153416**	512K 65536K
FOX32 Unit: 12	144.150MHz 144.325MHz	102_73181_10 102_73181_28	85mW V4.08	7.40V Idle Tx **6.20V** S0	40mA Idle Tx 159mA	144.325 / **W7P5QQV6** Off:41 ID: **191327**	512K 65536K

Print "fox_label_A_top.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.

Figure 22.7: 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.

Chapter 23

Informal Fox Hunt Procedures and Rules

General checklist for hunt preperation.

23.1 Prior to the Hunt

A day or two prior to the hunt.

Run the **fox_label** program to generate a new set of event labels, transmitter logsheet, and other cards that are useful for the hunt.

Note that the **fox_label** program generates more labels than you will need for the hunt, so don't print any more than you need. Details of the **fox_label** program are in section 16.1 starting on page 281.

23.1.1 Transmitter Time Update

Install an antenna or dummy load as you plug in each transmitter in turn to update the TOY clock. Use the **fox_simple** utility descripbed in section 12 on page 235.

Something along the lines of:

```
fox_simple -t10 -SFOX2X
```

The **-S** argument is unique for transmitters that have the FT232R installed.

Check battery voltage as you run the update. Installing the **TEST** jumper keeps the signon message from transmitting and should run the **STAT** command which reports current battery voltage.

You can use the preparation checklist (in section 22.7 on page 380) to record battery readings.

23.1.2 Labels

Files we will **not** use:

- fox_label_A_bot.pdf
- fox_label_B_power.pdf

Files we will use:

- fox_label_A_top.pdf
- fox_label_A_chk.pdf
- fox_label_B_cards.pdf
- fox_label_B_hunter.pdf
- fox_label_C_FOX21.pdf
  . . .
- fox_label_C_FOX32.pdf

fox_label_A_chk.pdf

This is our transmitter checklist.
Update the *fox_label.csv* file as you set the Toy CLOCK to reflect the current hardware configuration if it has changed.

You need to re-run the **fox_label** program after changes to the *fox_label.csv* file before printing any serialized labels. You need to re-run the **fox_label** program The **fox_label_A_chk.pdf** has the serial number information so you'll need a fresh copy of this for the hunt.

fox_label_A_bot.pdf

These are the FCC station identification labels that do **not** change. These labels should be permanantly affixed to the bottom side of the enclosure, the side the circuit board is fixed to.

Do **not** print these labels.

fox_label_A_top.pdf

These are the hunt serialization labels. The hunter needs visual access to this label to record the serial numbers on hi/her hunt card.

These are serialzed, so must be printed along with any other serialzed labels from the same ruin of the **fox_label** program.

fox_label_B_cards.pdf

These are the hunter log cards. The hunter records that serial numbers from the fox transmitter on this card during the hunt.

Print enough for the expected turnout.

These cards are **not** hunt specific. Save any leftover cards for the next hunt.

fox_label_B_hunter.pdf

These are the hunter registration cards.

Just before the start of the hunt, each participant records their callsign and starting tim on this card.

When they complete the hunt, record the finish time and the number of fox transmitters that were found.

These cards are also **not** hunt specific. Save any leftover cards for the next hunt.

fox_label_B_power.pdf

These are power labels for the RF amplifiers.

Do **not** print these labels.

fox_label_C_FOX*.pdf

These are the transmitter serialization cards. These are reprinted for each hunt with the unique serial number for the transmitter.

A stack of these are rubber-banded to the transmitter so the hunter can take one as a record of finding the tranmsitter.

These are serialzed, so must be printed along with any other serialzed labels from the same ruin of the **fox_label** program.

23.2 Receiver Preparation

Load your tracking receiver up with all of the operating frequencies that will be used during the hunt.

As an example, this is the authors setup for a Kenwood handheld:

Rx Channel	Rx Frequency	Description
10	144.150	Common Announce Frequency
11	144.225	FOX21..FOX26 Hunt Group
12	144.250	FOX33..FOX38 Hunt Group
13	144.285	FOX5..FOX8 (FOX20) Hunt Group
14	144.300	FOX20 Training
15	144.305	Raspberry-PI Transmitter
16	144.325	FOX27..FOX32 Hunt Group
17	144.565	WB6EYV MicroHunt Transmitter

23.3 Transmitter Preparation

Affix the *fox_label_A_top.pdf* labels to the corresponding transmitter. These are affixed to the removable cover of the transmitter.

Rubber-band the stack of *fox_label_C_FOX*.pdf* cards to the transmitter.

Verify power off!!!
Power switch flipped away from antenna conenctor.

You may, to verify battery condition, switch the transmitter on to hear the battery report on 144.150MHz. If the battery is above 7.2 volts, you probably have enough battery power left for the hunt. See the power plots in section 4.10.2 on page 61.

Again, verify power off when you're done!!!
Power switch flipped away from antenna conenctor.

23.4 Transmitter Deposit

This one is simple.
Set you handheld radio to 144.150MHz.
Find your *hiding* spot, keeping the bright orange antenna and the serialized label and serialized cards visible, switch the transmitter on and listen for the signon message as you place the transmitter.

The TOY clock keeps them all in sync, so no special handling required. An inadvertant bump of the power switch is remedied by simply turning it back on, it will be in sync with everyone by the next cycle.

23.5 Participant Signin

Give the participant one of the **fox_label_B_cards.pdf** cards to log their hunt and have the participant fill in the **fox_label_B_hunter.pdf** card with a name and callsign. Also log the starting time in the **Time OUT** field. Keep the **fox_label_B_hunter.pdf** card at the control area.

Remind them to take a **fox_label_C_FOX*.pdf** card from the transmitter or record the serial numbers on their log card. Only **one fox_label_C_FOX*.pdf** card please!

23.6 Active Hunt

You can monitor the transmitters on their assigned frequencies.

The **fox_label_A_chk.pdf** has the operating frequencies.

Only one transmitter should be active if you've correctly set up the sequences and the MOD schedules. When running a full hunt group, one transmitter in the group should be almost always be active. With a well built up sequence inter-message gaps should be short

Multiple transmitters active at one time indicates a problem with the TOY clock or an incorrect MOD configuration.

23.7 Hunt Completion

When your hunters return, record the current time in the **Time IN** field on the **fox_label_B_hunter.pdf** card.
Count collected cards and verify hand-written numbers with you master copy on the **fox_label_A_chk.pdf** sheet.

Calculate elapsed time and record in the **Time Delta** field.

23.8 Hunt Teardown

Collect all the transmitters.

No power switch rules, simply shut it off.

If you want/need to know if it's still working, turn it back on and tune to 144.150MHz for the signon message.

23.9 Awards Ceremony

May the best man win...

Sort the **fox_label_B_hunter.pdf** by the **Units Located** field.

Then, within **Units Located**, by **Time Delta** field.

Your event winner is on the top of the pile. Award the winner the melted Snickers bar.

23.10 Hunt Rules

Here are our basic rules:

23.10.1 Observe Venue Rules

We are guests in our hunt venue, be respectful of venue rules.

Transmitters will be accessible, such that you can easily reach a transmitter log card or see the label. The antenna may not be visible from your vehicle.

Your hunt organizers probably detest stickle burrs and ticks (not to mention lions, tighers, and bears).

Oh my!

23.10.2 Equipment

You have to walk around (or driver around) with your receiver and antenna system. Other than practical limits, we like to see what you bring to the hunt.

A reminder for the novice: the transmitter emits energy in the 2m spectrum **as well as the 575nm to 600nm spectrum**. Energy in the 575nm to 600nm spectrum is **strictly line-of-sight**. Energy in the 575nm to 600nm spectrum is obstructed by leaves, grass, paper, weeds and trees.

Fox Transmitter

The fox transmitter itself is housed in a Hammond 1599E enclosure. All of the existing fox transmitters are **black**. The dimensions are 6.69 in. by 3.37 in. They are 1.37 inches thick.

The transmitting antenna is bright orange!

Fox Transmitter Identification

Each transmitter has a permanent label on the bottom that identifies the equipment and the licensees authority to operate.

On the top of the enclosure, which should be visible to the hunter, is the hunt identification label and a small collection of *identity cards*.

Take on (and only one) of the *identity cards* to record your find. If none of the *identity cards* are left, record the serial numbers from the identification label. See examples in section 16.5 on page 287.

Fox Hunt Cards

Check Sheet
Hunt Labels
Transmitter *found* cards
Hunter Log Cards
Hunter In/Out record card

23.10.3 Checkin

The event is timed from your departure to your return. We're laid back for this event.

You should be given a log card, with room to record 9 transmitter ID/Code. Feel free to record on the backside of the card.

You will also fill out a hunter card with your name/call and time out. This card remains at the checkin desk. When you are finished, your finish time is recorded on the card along with you transmitter count.

23.10.4 Transmitters

Please do not switch the any of the transmitters off!

It makes them hard to find when the event organizers forget where they left them :-)

Should you bump the switch, no harm / no foul, simply move it back to the on position (towards the antenna connector) and it will take care of itself.

Please take only one identification card from the stack that is rubber banded to the transmitter. If none are left, record the **ID** and **Event Validation Code** from the transmitter label on your log card.

23.10.5 Hunt Groups

We have enough transmitters to operate multiple concurrent hunts. Each **hunt group** consists of up to six transmitters operating on a common frequency.

Additional **hunt group**s will operate on unique frequencies.

23.10.6 Team Hunt

If we have enough participants, why not!