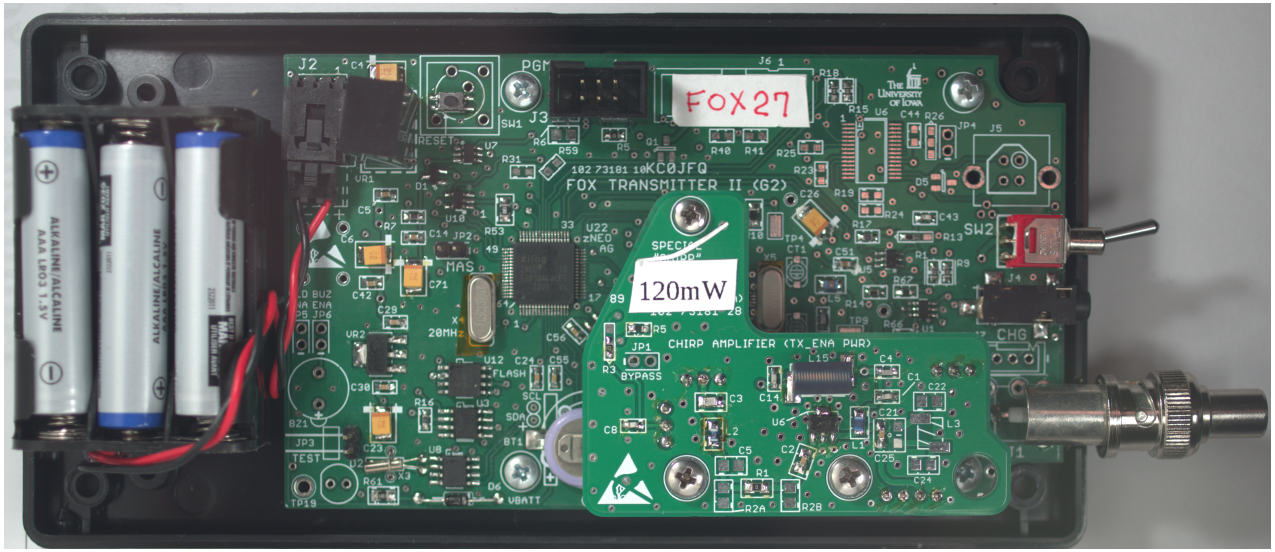


The ICARC FOX Transmitter

High Speed Binary Loader

KC0JFQ: William Robison

Job: fox˙present˙12
File: fox˙present˙12.tex

May 29, 2025

Iowa City Amateur Radio Club

Outline

Iowa City Amateur Radio Club

Overview

Why did Binary Loader arise?

Waveform memory is **LARGE**.

Sequence memory is none too tiny either...

command line InTeL HEX File record loader

accepts records one line at-a-time.

command line Echo a brief **status** message after each line.

STSxx,00 ...*

command line Echo a brief **ready** message after last line.

RDYxx,00 ...*

This message is intended to allow closed-loop operation

command line Running open-loop, this is slooooooooow

Status reporting is, perhaps, a bit verbose for closed-loop?

Iowa City Amateur Radio Club

Protocol Introduction

Command-line is **not binary.** We need two bytes to load 8 bits.

We need 76 characters to load a 32 byte *page* in FRAM/FLASH.
We can reduce this to 42 bytes by running a binary protocol!

Fixed length records

Reduce overhead in Fox Transmitter **ISR** (no length extraction)

Fixed format records

Reduce decoding overhead in Fox Transmitter (eliminate parsing)

Binary Data

Reduce size of packet (and time to send)

Single Character response to host (reduce host decode time)

ACK 0x06 indicates pass
NAK 0x15 indicates fail

Alternate bit rate (reduce channel active time)

115,200 is peak speed for 20MHz crystal
Reduce time to send packet

zNEO Limitation

zNEO Program Memory 128KB

- Version 4+ firmware almost fills up the 128KB flash

zNEO SRAM Memory 4KB

- Very tiny RAM!!!
- Allocate dynamic data on the stack
- Push static data into program flash

FRAM at least 256Kb (larger devices get expensive!)

- Operating *sequences*
  - This is where the operating *instructions* live.
- The 256Kb device stores 1024 commands (*instructions*)

FLASH more than 8Mb (larger devices are still dirt cheap!)

- Audio Waveforms
  - RIFF/WAVE file format (with header)
  - 8 bit mono unsigned data
  - RIFF/WAVE header provides sample rate and count
  - 8Mb is a bit over 4 minutes of 4KHz audio

Frame Format

SOH **0x01** 8 bits Start of Header

Notify ISR Start of a data packet. All other data patterns discarded.

Length 16 bits

Data packet length (always 32) Length of ZERO is special!

Address 32 bits

Record Address (byte address)

STX **0x02** 8 bits Start of Text

Protocol requirement

Data Packet 160 bits (32 octets) of (binary) data

8 bit (clean) channel required!

ETX **0x03** 8 bits End of Text

Protocol requirement

CKS 8 bits Packet Checksum

End-Off 8 bit sum of entire packet (results in zero)

EOT **0x04** 8 bits End of Transmission

Protocol requirement

Iowa City Amateur Radio Club

Fox Transmitter Binary Commands

Normal Command activates binary mode of operation
Normal **sts46,00*** message sent by Fox Transmitter before transition
Normal **STS46,00*** and **RDY46,00*** messages are sent
by Fox Transmitter after return to normal mode

H56K PROG Set bit rate to 57,600 b/S

Load directed to FRAM device
Download data overwrites existing data (ERAS not needed)

H56K WAVE Set bit rate to 57,600 b/S

Load directed to FLASH device
FLASH device **must** be in erased state!

H115 PROG Set bit rate to 115,200 b/S

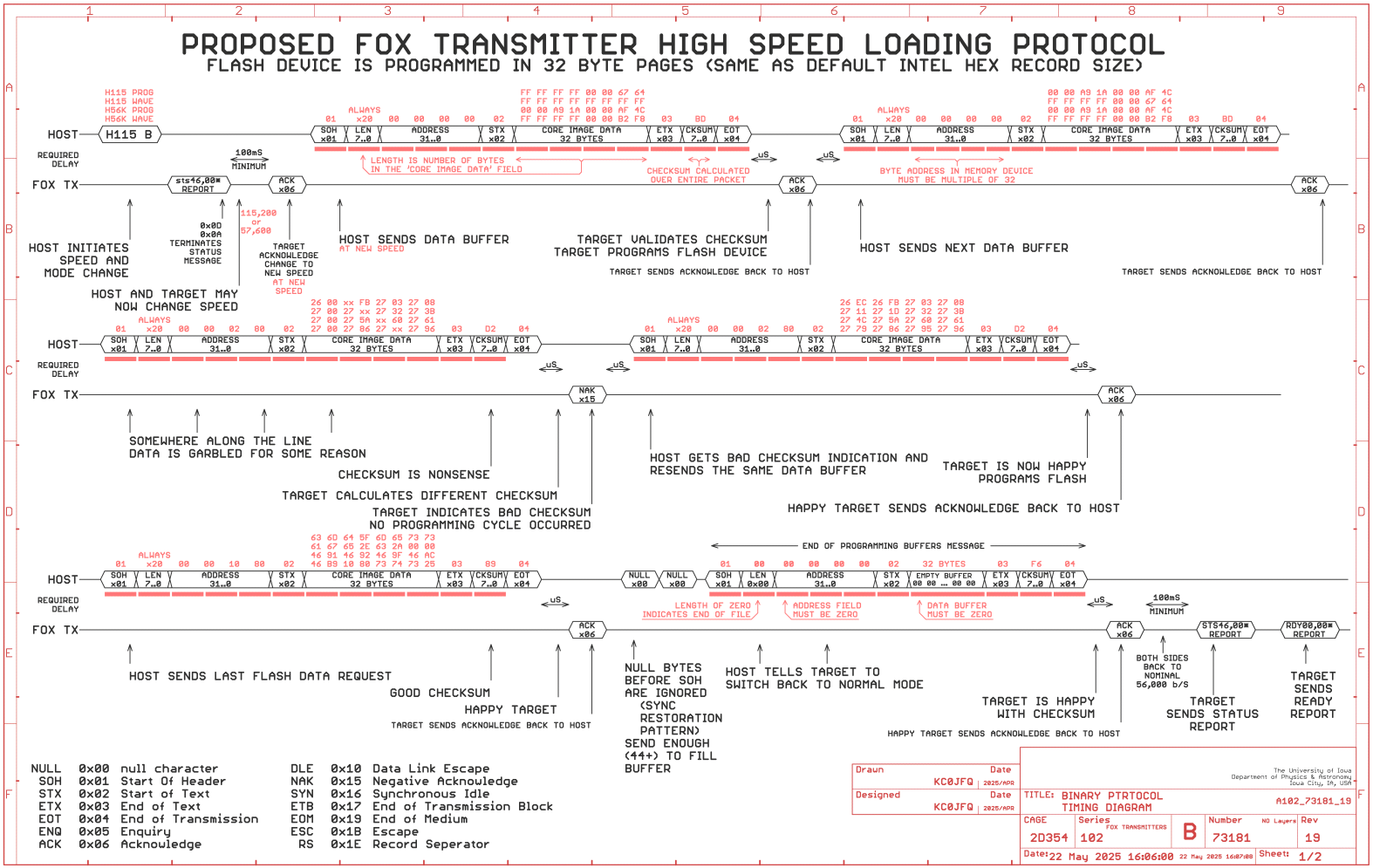
Load directed to FRAM device
Download data overwrites existing data (ERAS not needed)

H115 WAVE Set bit rate to 115,200 b/S

Load directed to FLASH device
FLASH device **must** be in erased state!

Iowa City Amateur Radio Club

Protocol Drawing



Iowa City Amateur Radio Club

Processing Overlap

Sorry, no overlap. We can't receive data as we program the last buffer.
- All because of the FLASH memory device

Validate the checksum and abort with NAK (0x15) if bad
- Host can re-transmit, abort, or continue on, Fox Transmitter doesn't care

If checksum is good, program the target device
- Translate byte-address to record number (shift right 5 bits)
- Poll for completion (on FLASH device) This is what prohibits overlap!
- Return ACK (0x06) if successful, NAK (0x15) if not.

Special case for length and address set to zero.
- End-of-File indicator. We are **done**!
- Return ACK Checksum validation already handled
- Switch back to normal bit rate (57,600 b/S)
- Delay around 100mS to give host time to switch
- Format the **sts46,00*** message

Exit through the normal command processing path
- This changes the **sts46,00*** message to **STS46,00*** and sends it to the host
- This generates the **RDY46,00*** message with a time report

Iowa City Amateur Radio Club

Other Commands

A few commands to manage download operations.

HERA ALL Erase the FLASH device

Large devices are sl*ooooo*w, some take 60 seconds! do **NOT** remove power!

The FRAM device is RAM, doesn't need to be pre-cleared.

HEND Test *FLASH busy* and find first free page of FLASH memory

Use this to verify that the erase completed

This command gets *annoyed* if the FLASH device is busy do **NOT** remove power!

EDMP ID= Dump the ID records in FRAM

The processing utilities generate these records and place them into the FRAM load file. These are not needed by the Fox Transmitter.

STAT System Status

Insepct all the system settings. Operating frequency, in particular!

ONCE S0= Test the S0 sequence

This runs through the named sequence *(S0= ... S9=)*

Use to verify audio data loaded.

Listen on the correct frequency!!!

Fox Transmitter host utilities

halo_term Simple terminal emulator

Use this utility to interact with the Fox Transmitter to run the commands seen on the previous page

fox_simple command line loader utility

This may be used to download FRAM and FLASH memory using the commands implemented in the Fox Transmitter.
No protocol, plain text. Plays well with the **halo_term** utility

fox_binary binary loader utility

This may be used to download FRAM and FLASH memory using binary protocol. The **halo_term** utility messes everything up!!!

fox_audio WAV file to InTeL HEX records reformatter

Converts WAV file to InTeL HEX records. This utility deals with placing the wave files in ascending locations in memory.

Iowa City Amateur Radio Club

Edit Sample

First Line

Indented Line

Iowa City Amateur Radio Club

Scary Notes for the Presenter

These are my crib notes. I sure hop I rememberd to print them off and bring them along...

Overview

Using **fox_simple** to load a 1/2MB file (4Mb) was taking forever. Up past an hour!

Tired of that shit, so make a faster protocol!

Didn't bother with operating cloddes-loop using the textbfRDY,00,00* message as we are still stuck with the text based Intel HEX format.

Straight to implementing a 8-bit clean protocol to load the FLASH device. The FLASH device must be page programmable. No longer support the AAI (auto address increment) devices that program 16 bits at once.

So lets get on with it!

Protocol Introduction

The command line is tailored for human interaction. I wanted to be able to test and control the Fox Transmitter using a terminal emulator.

A sequence is just a list of stored commands that are fed into the command parser.

Intel HEX files start with a colon (:) which the command parser recgonizes, so you can simply shove Intel HEX records at the command parser and it will send them to the FLASH device.

The Intel HEX record has a means of dealing with a 32 bit address space, so we just take the collected audio image and produce a file with Intel HEX records.

We can then shove these Intel HEX record down the command parsers throat using the **fox_simple** utility.

BUT it's dog slow. Well over an hour to load a one half Mega BYTE file. That's 4 MEGA BITS.

So lets define a simple protocol to load the flash.

Simple as far as the zNEO is concerned as it has limited memory (program flash and SRAM). We also have to limit the instruction pathlength inside the zNEO ISR to be able to run at 115,200 bits/Sec.

zNEO uses a 20MHz crystal which limits the bit rate to 115,200. Above that speed we can't get a good divisor. Using an 18.432MHz crystal might address that, but the keeping wall clock time becomes a problem.

The ISR, without some very careful tuning, also tops out at 115,200.

So, use a fixed length record to simplify the ISR. It doesn't need to dig in the data record to find length data. A bad length field can't cause problems. The instruction pathlength in the ISR is reduced.

Using a fixed format makes decoding a bit easier in the zNEO. This speeds thing up slightly but more importantly keeps the footprint in program memory down.

The purpose of using binary data is to cut down on the number of bytes we have to move from host to target.

Response to host must be generated in the Fox Transmitter so keep that simple as well. Limit to a single character GOOD/BAD response.

The zNEO UART has a simple clock generator. It can be altered by writing a 16 bit register. So we can operate at 56K or 115K with no adverse impact.

The binary loader deals with a single *FILE* and returns to the command parser. The command calls out the bit rate and the target device. We then load data, 32 bytes at a time, until host tells Fox Transmitter that host is done.

zNEO Limitation

The zNEO started out looking like the program flash would be more than adequate. Well, we disavowed that notion. It's almost full. Seems to be about 10K bytes left.

The SRAM is also limited, stingy 4KB. UGGH!

We can't be careless with data buffers so there is quite a bit of buffer sharing that occurs with UART data.

Local data for almost all subroutines is allocated on the stack by the compiler (plain old C). We expend considerable effort to avoid static variables.

Care is taken to promote static strings to program flash. The compiler has a mechanism to handle this. Typing **HELP** produces considerable text that has been promoted to program flash.

We never attempted to store operating instructions (what are typically called sequence commands) in zNEO memory. It's too small and to easy to corrupt the instruction area of program flash.

This gives rise to the FRAM. The operating *personality* lives in this FRAM. The FRAM is persistent, so it lives on without power, and can be written at wire speed.

Writing to the FRAM bypasses the file system so it provides a fast and convenient means of updating the operating sequence.

The FLASH device is there for audio waveforms, so the target size is megabytes. An FRAM device this size is $25 to $50 so a FLASH was added to the 102-73181-10 revision. A 8Mb FLASH device sells for about $1 and a 64Mb device comess in at $1.75, so Bob's your uncle.

Where the FRAM doesn't require an erase prior to writing, the FLASH must be erased before it is re-written. The erase operation can be slow (some of the larger devices can take a minute to erase) so it's not part of the protocol.

All we can do using the binary loader is write!

The FLASH is loaded using Intel HEX records that are produce using the

B

B

B

B

B

B

B