Using Microprocessors
The ICARC Fox Hunt Transmitter Family

Presentation by KC0JFQ

William Robison

March 18, 2025

A microprocessor application

http://n952.ooguy.com/HamRDF/index.html
http://n952.ooguy.com/eagle/index.html

Job: fox˙present˙9
File: fox˙present˙9.tex

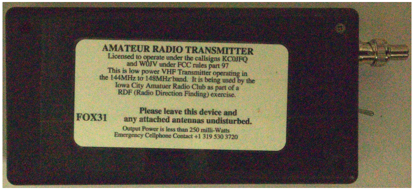
Table of Contents

What can we do with single-chip computer?

Single Chip microprocessors are in everything these days

A hands-on project for weekend entertainment:
A bit of kit for radio fox-hunting.

Can we come up with a micro processor system project to control a radio transmitter for amateur hunters?



AMATEUR RADIO TRANSMITTER
Licensed to operate under the callsigns KC0JFQ
and W0JV under FCC rules part 97
This is low power VHF Transmitter operating in
the 144MHz to 148MHz band. It is being used by the
Iowa City Amatuer Radio Club as part of a
RDF (Radio Direction Finding) exercise.

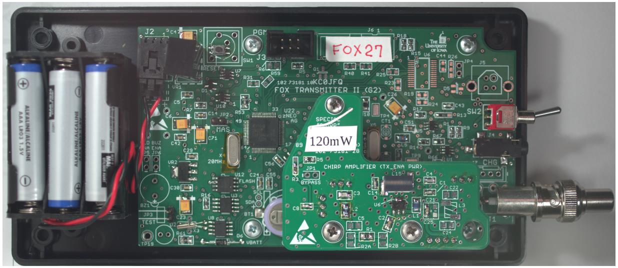
FOX31

**Please leave this device and
any attached antennas undisturbed.**
Output Power is less than 250 milli-Watts
Emergency Cellphone Contact +1 319 530 3720

What can we make with single-chip computer?

Wide Universe of **SOC** to choose frrom

SOC = System On Chip



Raspberry PI

Raspberry-PI Zero

ARM

Arduino

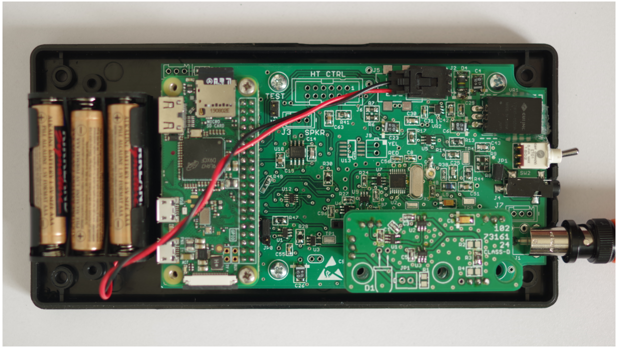
PIC12/PIC14

ZiLOG zNEO

Iowa City Amateur Radio Club

Raspberry-PI

The Raspberry-PI Zero-W is small

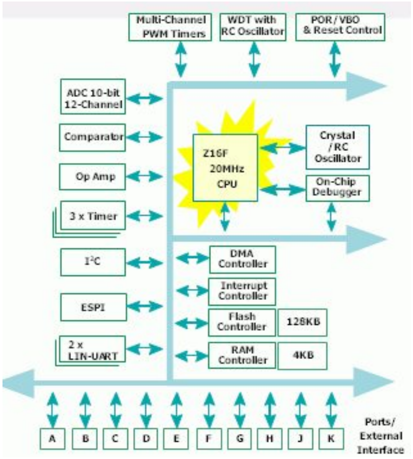


Project built with the Raspberry-PI-ZERO fits in our box!

The -PI is a power PIG, one fox-hunt per battery set

Prone to SD Card corruption requiring re-load

Typical SOC: **ZiLOG ZNEO**



Hardware

- 128KB Program Flash
- 4KB Data RAM
- Large set of peripherals
- Simple Flash programming interface

Everything we need/want in one package

What's missing

What else do we need

Hardware

Circuit board, somewhere to put the parts

Power, battery or wall-wart

Software

Program to run the system

Instructions loaded into the Flash Memory

Temporary *stuff* in RAM

The EXAMPLE Project

A *FOX HUNT* transmitter

What is FOX HUNT ?

Someone hides the transmitters

Everyone else tries to find them

Why do we use the SOC

FCC rules dictate *station identification*

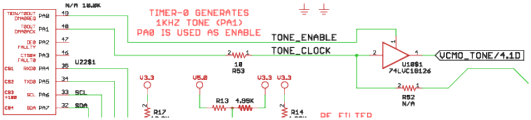
Five or six transmitters time-multiplexed on the same frequency

The transmitter can talk!

Audio

Control System: **Audio**

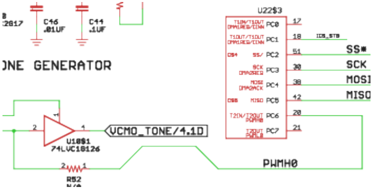
Your introduction to reading schematics



Morse Code Generator

Simple square wave

That VCMO_TONE goes off to drive the radio



Voice Generator

PWM

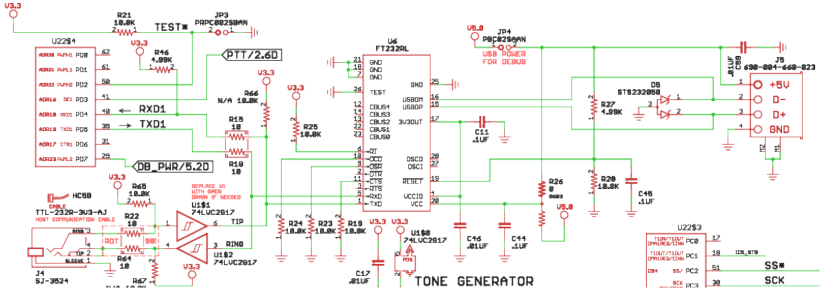
That PWMH0 goes off to drive the radio

Communications

Control System: **Serial Ports**

Your parents (grandparents?) did computer *surfing* using a MODEM

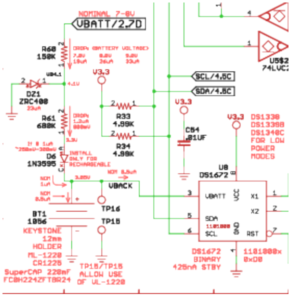
MODEM connects with 2 data lines, one each direction



TOY Clock

Control System: **Time Of Year clock**

Why we want time



Five or six systems running at once

All sharing the same channel

Take turns talking *on the air*

First talks at 10:00:00 for a minute

Second talks at 10:01:00

Third talks at 10:02:00

Rinse and Repeat!

More Memory

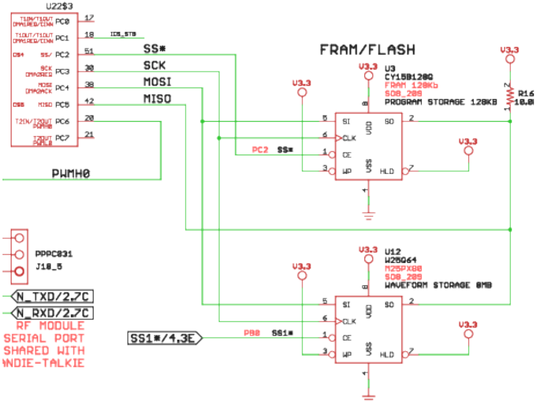
Control System: **External Memory**

Program Flash needs special programming hardware

Software knows how to write and erase FRAM and FLASH

FRAM has*FOX HUNT* **plan** (operating commands)

FLASH has audio waveform data (i.e. voice clips)



Other Raspberry-PI Projects

Raspberry-PI Projects that exist

102-73161-0 Raspberry-PI Fox Transmitter
  Talking system for FOX Hunting

02-73161-0 Raspberry-PI Talky Toaster
  Uses that same board
  Adds LEDs, Switch, Light Sensor, R/C Servo Control

102-73173 Plant Killer Base
  Houseplant Monitor (monitor soil moisture)
  Use the WiFi features of the Raspberry-PI to send soil moisture reports

102-73173 CI-V Aux Control
  Switches to control Ham Radio operation
  CI-V is the control interface for ICOM radios

102-73209-0 NMEA Logger
  Capture data from fishfinder (lat/long/depth)

Raspberry-PI Projects, artwork only

102-73171 LED Display Control
  Incomplete project to control large 7-segment display

Secret Content

Pay no attention to the man behind the curtain

End of the line We are **done here**

Go away Get Back

Dragons be here

You are not permitted to look beyond here

Scary Notes for the Presenter

Notes for the terminally forgeful

Here we go again!
-Dolly Parton

What can we do with single-chip computer?

Slide: 3

This is a project that makes use of a SOC, a **System on Chip microprocessor**.

Most of the functional *stuff* is inside the SOC.

We'll use the **ICARC Fox Transmitter** as an example.

It started out using a single-chip processor, moved to a Raspberry-PI and back to the SOC.

The move away from the -PI was a result of issues that crop up when you have a General-Purpose O/S (Linux) in control of your project. In particular when the OFF-ON switch can remove power without shutting down the file system.

What can we make with single-chip computer?

Slide: 4

The hardware presented here is a project started in 2018 make a very flexible fox transmitter for the Iowa City Amateur Radio Club.

FOX Hunting where the *hunters* attempt to find a group of hidden transmitters.

Each one talks for minute, in turn. After they all get a chance to *talk*, the first one goes again and it all repeats.

What processor to choose to implement this fox transmitter system?

PIC12/PIC14

- Dirt Cheap
- **DOWNSIDE:**
  - Most are pure Harvard Architecture
  - Most are too small of a memory footprint
  - Vendor tools ($$$)
  - Not compiler friendly!

Arduino

- AVR-DU Commonly used.
- Linux toolchain (makes author happy)
- **DOWNSIDE:**
  - Unfamiliar to author
  - Limited Program Memory (32KB visible), slightly bigger RAM (8KB)
  - Harvard Architecture

ARM

  - Newer Arduino boards use ARM
  - STMicro has huge pile of development boards
- Large memory space, both Flash and RAM
- Fast clock speed
- Development tools readily available and low cost
- Linux toolchain
- **DOWNSIDE:**
  - Unfamiliar to author

Raspberry PI

- **Very** large memory footprint
- Runs Linux
- Easy to program and debug
- Has WiFi to update things
- **DOWNSIDE:**
  - Power PIG (when we're running on batteries)
  - File System corruption when powered down incorrectly.

ZiLOG zNEO

- Reasonable memory footprint
- Mixed VonNeuman/Harvard Architecture
- Plays well with the "C" language
- **DOWNSIDE:**
  - Package availability
  - ZiLOG seems to be coasting into oblivion

There are little -PI (Raspberry-PI Zero) and big -PI models (Raspberry-PI-3, Raspberry-PI-4) examples on the desk.

Fox transmitter is the Raspberry-PI Zero and it is powered by 6 AAA batteries (these are the little ones!)

The PI runs Linux. After using it a bit, it will probably seem easier to work with than a Windows based system.

Most everything is a shell script or Python. Some low-level functions written in **C** like the code that manages the RF synthesizer.

No screen (rather obvious) so don't want to bother with any graphical software. Writing in Python and **C** so we simply ignore any graphical stuff, the Fox Transmitter application is all procedural.

The -PI is still generating a graphical user interface, but we're not using it (-PI Zero has HDMI port for screen!). The shell-script that controls it starts when the -PI boots up.

Entry in the startup scripts to run the Fox Transmitter application. I googled it after writing and testing the application code and promptly forgot what I did. I can always hook up a monitor or use a remote desktop (like VNC or something similar) to go in and look at the scripts and code that runs it all.

The SD card is backed-up once it's all up and running so recovery from a corrupted SD card needs only to restore the backup image to the SD card.

The backup is a raw dump of the card, not a file system dump. Use **dd** to save and restore.

This **raw dump** is not compressed. takes up as much space as the size of the SD card.

Since theres no *control* over the system, other than the power switch, when out in the field, it's not difficult to hose the SD Card. ARRGH!!!

Also, the ARM chip and the SD card aren't all that fast, so time from power-on to running the application is slooooooow.

The **Talky Toaster** project can illustrate that.

The PI suck the life out of the battery. It's only goos for one fox hunt before uit needs to be replaced.

The Raspberry-PI is still way less power than most things you'll work with on the desktop. The PI-4 and PI-5 get by on a 10W or 15W supply.

T

A

Raspberry-PI Slide 5

This came about to be able to do voice, it talks.

A lot less power than the Raspberry-PI-3 or Raspberry-PI-4 and a lot smaller so it fit in the target enclosure.

-PI has a **PWM controller** so it has audio capability *built-in*.

Pretty much same hardware as the zNEO version.

Had to add external A/D as -PI doesn't have this...

ZiLOG zNEO Slide: 6

This device used extensively for previous projects so author is familiar with these devices. Programming tools readily available from ZiLOG. No-cost compiler, low cost USB device programmer.

Used extensivly at VanAllen Hall for GSE.

Device programming simple enough for building your own programmer. Which, of course, was done to solve an O/S issue. Also required programming software on the Linux host.

Compiler tools can be operated under Linux. Author doesn't use Windows, so tools must be able to execute in the Linux environment. WINE emulator to the rescue.

The peripherals in the block diagram are typical of what you gind in most SOC chips.

Raspberry-PI processor chip covers slightly different applications, so the collection of peripherals is different, The A/D function is missing.

What's missing from the zNEO? Slide: 7

Not much.

The zNEO is a bit shy on memory so more is needed, but most of the rest is right on the chip. Extra external memory solves that problem.

Everything is surface mount these days so it's difficult to bread-board. This project was tested on a circuit board. Some *haywires* were required on the early boards.

zNEO has fairly broad selection on on-chip peripherals. Can do things with just the zNEO chip, a crystal, and some buffering.

The Fox Transmitter project doesn't use too many of the zNEO peripherals or package pins.

The software loaded into the zNEO program flash controls this, of course.

It implements a simple verb/noun processor to implement the fox transmitter function or personality.

The commands that implement the **personality** of the Fox Transmitter are all stored in the external memory. The audio waveforms are also stored externally.

The EXAMPLE Project Slide: 8

What is shown here is a Ham Radio project started in 2018 to develop a feature rich fox transmitter for ICARC fox hunts. Six transmitters are place in an area to be located by the hunters.

They all transmit on the same frequency. Each one gets a minute to transmit and then goes quiet for five. Takes a bit of discipline to keep your head in the game and find them all.

BUT, there's more. Two more hunt groups.

Software is all in the **C** language.

It requires a bit more that 120KB of the (128KB) program flash in the zNEO.

The 4K RAM in the zNEO is barely adequate.

One must be careful when creating code to keep things out of RAM that don't need to be there. Must use many *vendor specific* extensions in the source code to get everything to the area of memory where it needs to be.

Load map from the latest build.

```
Space                   Base         Top       Size      Used    Unused
------------------ ----------- ----------- --------- --------- ---------
ROM                T:   0000   T:   67C5      8000H     67C6H     183AH
                                          (   32768     26566      6202)
EROM               C:008000    C:01DEF9      18000H    15EF9H     2107H
                                          (   98304     89849      8455)
RAM                R:FFB000    R:FFB8B0      1000H      8B1H      74FH
                                          (    4096      2225      1871)
```

The application does **not** use heap memory (i.e. no malloc() calls).

It does use lots of stack space for dynamically allocated variables.

ROM and EROM are both in the 128KB program flash (0x000000 to 0x01FFFF).

Timers

- 100Hz system heartbeat and system time
- Tone generator for morse code
- Times morse code (interrupt rate is *DIT*)

- UART (Universal Asynchronous Receive Transmit)
  - Host control port
  - Radio Module control port
- PWM Controller (Pulse Width Modulator)
  - Audio out to radio circuit
- SPI (Serial Peripheral Interface)
  - FRAM memory device
  - FLASH memory device
- I2C (Inter-Integrated-Circuit bus)
  - TOY clock
  - SI5351 RF synthesizer (radio carrier generator)
- A/D (Analog to Digital)
  - Battery Voltage
  - Battery Current
  - 5 Volt Regulator

- GPIO (General Purpose Input/Output)
  - Power control bits
  - Tone enable bit (**TONE_ENABLE**)
  - RF Enable bit
  - Jumper Monitors
  - PTT control (Push-to-Talk)
    - Makes the radio transmit
- DBG (Programming/Debug Port)
  - Update zNEO program flash

Audio Slide: 9

This project is simple radio, so it has to make some noise. At a minimum, it has to do morse code to operate within FCC rules. The transmitter has to identify itself at the end of every transmission (morse code is sufficient).

It woudl be really neat if it could talk as well. As things turn out, we can use a PWM controller to make audio. Fortunately for us, the zNEO has a PWM controller.

To generate4 morse code, we use one of the zNEO timers to generate a signal at aq frequency you can hear. For our fox transmitter we go between about 300Hz to about 2KHz. The frequency limts are imposed by the way the radio works, not by how the zNEO works.

> SO, we make the audio tone (**TONE_CLOCK**), and then turn it on and off (**TONE_ENABLE**). It could have been done without the external on-off switch, but the the external on-off switch makes the software a bit easier and makes another feature work.

To make it talk, we need a DAC (digital to analog converter), which is usually a bit more complicated than what we do here. For our fox transmitter project we apply some signal theory and make the DAC out of a single on-off bit (the **PWMH0** signal).

> Turn **PWMH0** on and off really fast (like at 80KHz) and vary the on/off ratio. Filter that and you get useable audio (this is how lamp dimmers work).

The volume of data needed to store the audio is way beyond what the zNEO can store internally. Think about how big an old-fashioned CD is and how much audio it stores.

> CD stores 600MB of data and that's about 60 minutes of audio. We don't do stereo, we use smaller samples (8bit vs. 16 bit) and we digitize slower (5KHz vs. 44KHz),
>
> Even so, 60 seconds of audio at our 5KHz sample rate ends up requiring 300KB to store it.

For now, just realize we store all this audio data in an external memory that we'll talk about in a bit.

Communications Slide: 10

Here I use the work *Communications* to describe the data communications link to the desktop computer used to load the operating commands and the audio data into the external memory devices.

Why such ancient technology, why not something new and fancy and fast?

Software and Hardware!

Serial ports are ubiquitous (that means they're everywhere and I have to use Google to spell ubiquitous correctly).

The hardware is easy (and cheap) to implement.

Drivers exist everywhere.

Many USB things need special drivers. That's probably OK for high-volume things, but for us making a small project like this, dealing with drivers is far beyond our capabilities.

Before each hunt we update the clock in the fox transmitter and check on the battery condition.

All the computer-stuff is done before so when the hunt setup starts, all we do is find a hiding spot, turn the transmitter on, listen for it to tell us its alive. The rest of the transmitters all get dropped the same way.

When we first comission the transmitter we have to load the operating commands and the audio waveforms.

The operating commands identify the transmitter with a callsign (like KRNA or KCJJ) and a nickname. It also define the message traffic that gets sent and the schedule it will run on.

Basically, these commands define the operating personality of the fox transmitter.

This set of operating commands and the audio waveforms usually don't change, but the zNEO software knows how to erase the external memory devices so we can start over with something new.

TOY Clock Slide: 11

Remember earlier I said that there are five or six transmitters all operating on the same frequency?

The TOY clock (Time Of Year) is just like the clock chip in your desktop of laptop. You set it and it does a mediocre job of keeping track of time. We don't have an internet connection (of course) so we can't be in regular contact with a time server to keep our clock set.

When the fox transmitter is turned on, it read the time from the TOY clock to set the system time (same thing occurs on your desktop or laptop). From then on the zNEO keeps track of time, updating its time number 100 time per cecond.

The fox transmitter, then, schedules activities based on the current time. So for our six unit hunt, the fox transmitter turns on every six minutes.

The second unit fires up right after the first is done. It knows what time it is, it can't hear any of the other transmitters. So when the correct time comes up, it transmits.

Scheduling Discussion

This is a bit involved, but if you're interested we can go into some detail here.

The zNEO software keeps time in a 32 bit seconds field and an 8 bit sub-seconds field. Sub-seconds are incremented every 10mS and they roll-over at 100. Sub-seconds roll-over increments the seconds field.

So zNEO is truckin down the road keeping track of time all by itself after getting the correct time from the TOY clock at the start of it all.

The schedule is defined in terms of a cycle **period** and and **offset** within the period. Both of these numbers are expressed in seconds.

To find out something about *now*, take the system time and divide it by the **period**, ignoring quotient but keeping the remainder. When the remainder matches the **offset** value, that means it time to send the message.

Each transmitter in the group runs with the same **period** and the each have a unique **offset**. So for a six minute **period**, the **offset**s will be 0, 1, 2, 3,4, and 5.

More Memory Slide: 12

And then we can talk about the extra memory needed to store the operating commands and the audio wqaveform data.

There are two external memory chips, U3 and U12. The zNEO is on the left, U22.

The chps are connected to the zNEO with a serial bus so the speed with which we read these memory chips is relatively slow.

The FRAM, U3, is non-volatile read-write memory. It doesn't need to be erased before new data can be written. This makes testing a bit easier.

> The non-volatile part keeps the contents intact when there is no battery powering the system.

The FLASH, U12, is also non-voltile, but it takes forever to erase the chip when you need to load different data into it.

The larger device take up to 60 seconds to erase.

The FLASH is used to store waveform data for making audio so it needs to be much larger than the FRAM used to hold commands. The audio waveforms are assembled on the host machine and loaded into the Flash.

I use a cheesy USB microphone to collect voice data and then reprocess it to get it into the format required by the fox transmitter. Recall the small sample size and low sample rate talked about earlier.

The audio file are large and slow to load (the loader in the zNEO isn't optimized for speed). The idea is that you build the audio once and load it once. Erasing the flash should be infrequent.

We can load additional audio clips after existing data, you just can't rewrite a memory location once you've written it. Flash memorry is programmed to a '0' and erased to all '1's.

Secret Content Slide: 14

Discussion

This

I

J

K