

W1MJ FOX2

Automated Transmitter for Hidden Transmitter Hunts

Updated 9/22/2023
Eliot Mayer, W1MJ
fox@w1mj.com

Contents

Introduction.....	2
Design Overview	3
Assembly	4
Circuit Board Assembly.....	4
Software Installation.....	7
Battery Eliminator Modifications (Optional)	8
Antenna Assembly.....	9
Circuit Board Enclosure (Optional)	9
Final Assembly	10
Message Creation & Installation	13
Configuration	14
Deployment	15
Fox Circuit Board Design.....	16
Pico Power Supply.....	16
Transmit Audio Interface	17
PTT Circuit	17
Receive Audio Detection Circuit	17
Battery Monitor	17
PC Board.....	17
Software Design Overview	18
Choice of Programming Language	18
Pico Storage Usage.....	19
Use of Artificial Intelligence	19
CircuitPython Modules	20
Program Outline.....	21
Software Design Details	21
Time.....	21
MP3 Decoding	22
PWM Output.....	22
Analog Input	24
Software Development Tools	24

Introduction

The W1MJ FOX2 is an automated transmitter designed for use in hidden transmitter hunts. These hunts are also known in the amateur radio community as fox huntsⁱ. The fox plays pre-recorded audio messages at regular intervals using a low-cost VHF/UHF FM transceiver. FOX2 is an update to the original W1MJ fox, with the following features:

- Transmits a sequence of mp3 message files, one per minute (configurable), then repeats
- Has two modes of operation, each allowing longer deployment between battery charges than the original design (which ran 24/7):
 - Scheduled Mode: Limits transmissions to scheduled daily hours
 - On-Demand Mode: Transmits upon request for 1 hour (configurable); activated by audio on the fox frequency.
- Periodically announces the battery voltage.
- Stops transmitting when the battery voltage is critically low.

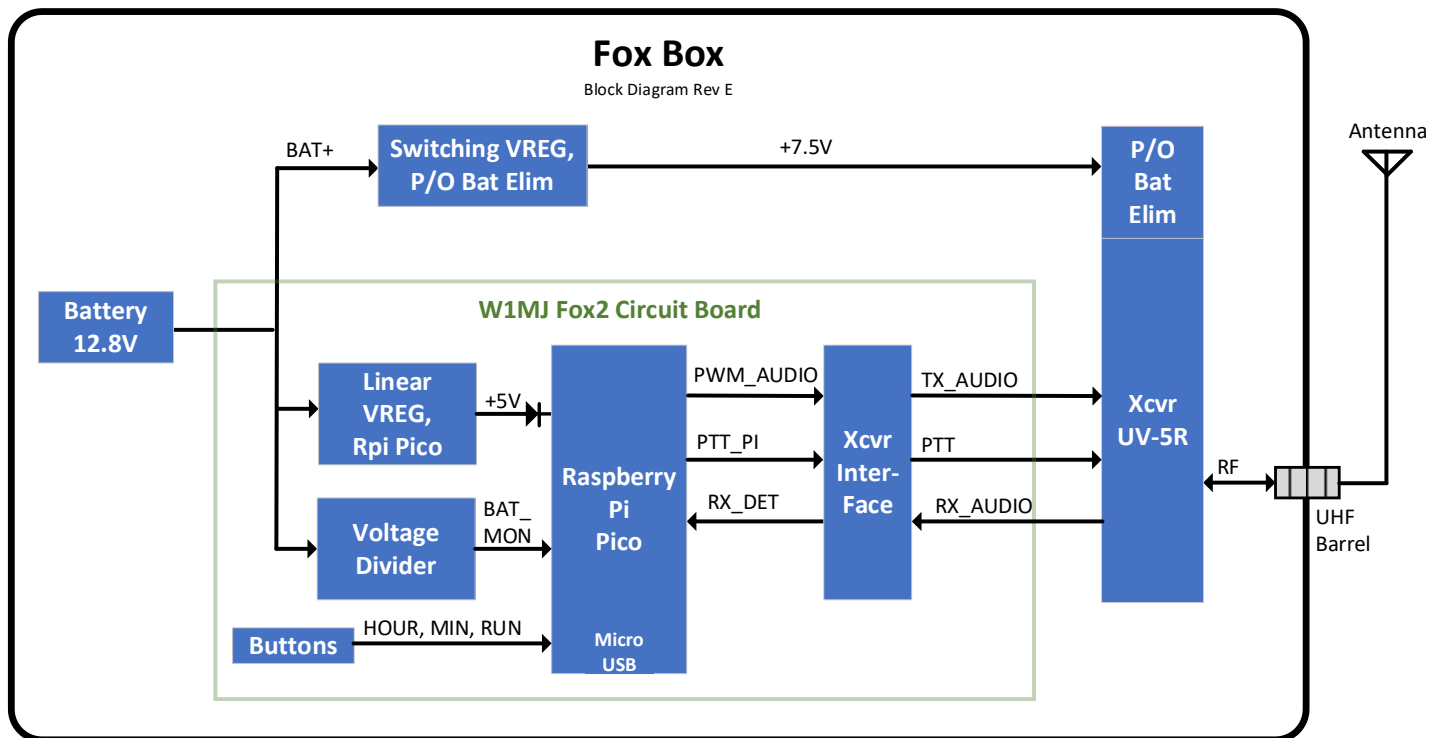
This article provides:

- Information for building, configuring, and deploying a W1MJ FOX2.
- Description of the fox hardware and software design

Design Overview

Figure 1 shows a block diagram of the fox.

Figure 1 - Fox Block Diagram



The main components are listed in Table 1.

Table 1 - Main Components

Component	Manufacturer	Part Number	Cost *	Comments
Battery, 12.8V 30AH LiFePO4	Bioenno	BLF-1230A	\$ 280.00	Smaller battery can be used if operating xcvr at low power or for shorter duration.
Transceiver, VHF/UHF FM, 4-watts	Baofeng	UV-5R	\$ 25.85	
Battery Eliminator	Baofeng	BL-5	\$ 18.49	
Fox2 Circuit Board	Homebrew	N/A	\$ 33.86	See Circuit Board section
Case	Pelican	1200 Black	\$ 54.95	
J-Pole Antenna	Homebrew		\$0.00	See Antenna section
UHF Barrel	From Stock			
Cables	From Stock			See Final Assembly section

* The costs are shown for my fox. Some prices may have gone up since then, but the transceiver price has amazingly continued to drop.

While the battery is pricey, mine has multiple uses, such as supplying emergency power for other amateur radio equipment and 12-volt gadgets. The second most expensive item is the case, but

Pelican has a good reputation, and this case has protected the fox through multiple deployments in varying weather conditions.

The transceiver, battery eliminator, and case were purchased from Amazon. The battery was purchased from Powerwerx. All the components for the circuit board, including the PC board, were purchased from Digi-Key.

The transceiver was selected for:

- low price,
- availability of “Battery Eliminator” to power radio from an external power source, and
- Interfacing information available online.

The Raspberry Pi Pico was selected because:

- very little support circuitry is needed to implement all the desired features, and
- programming was fairly easy using CircuitPython.

For design details, see the Fox Circuit Board Design and Software Design sections.

Assembly

Circuit Board Assembly

Assembly using ESD precautions is advised due to the CMOS circuitry in the Pico.

A prototyping board would probably suffice for this fox circuitry. However, to reinforce the [Altium Designer](#) training that I received at work, I opted to create a custom printed circuit board. I had a limited quantity of extra bare boards at the time of writing this article; if interested, contact me at fox@w1mj.com.

The Bill of Material (parts list) is detailed in Table 2. All these parts are available from Digi-Key, though I purchased the Pico from Adafruit with the header pins installed ([Adafruit Product ID 5525](#)).

Figure 2 - Raspberry Pi Pico with Header Pins

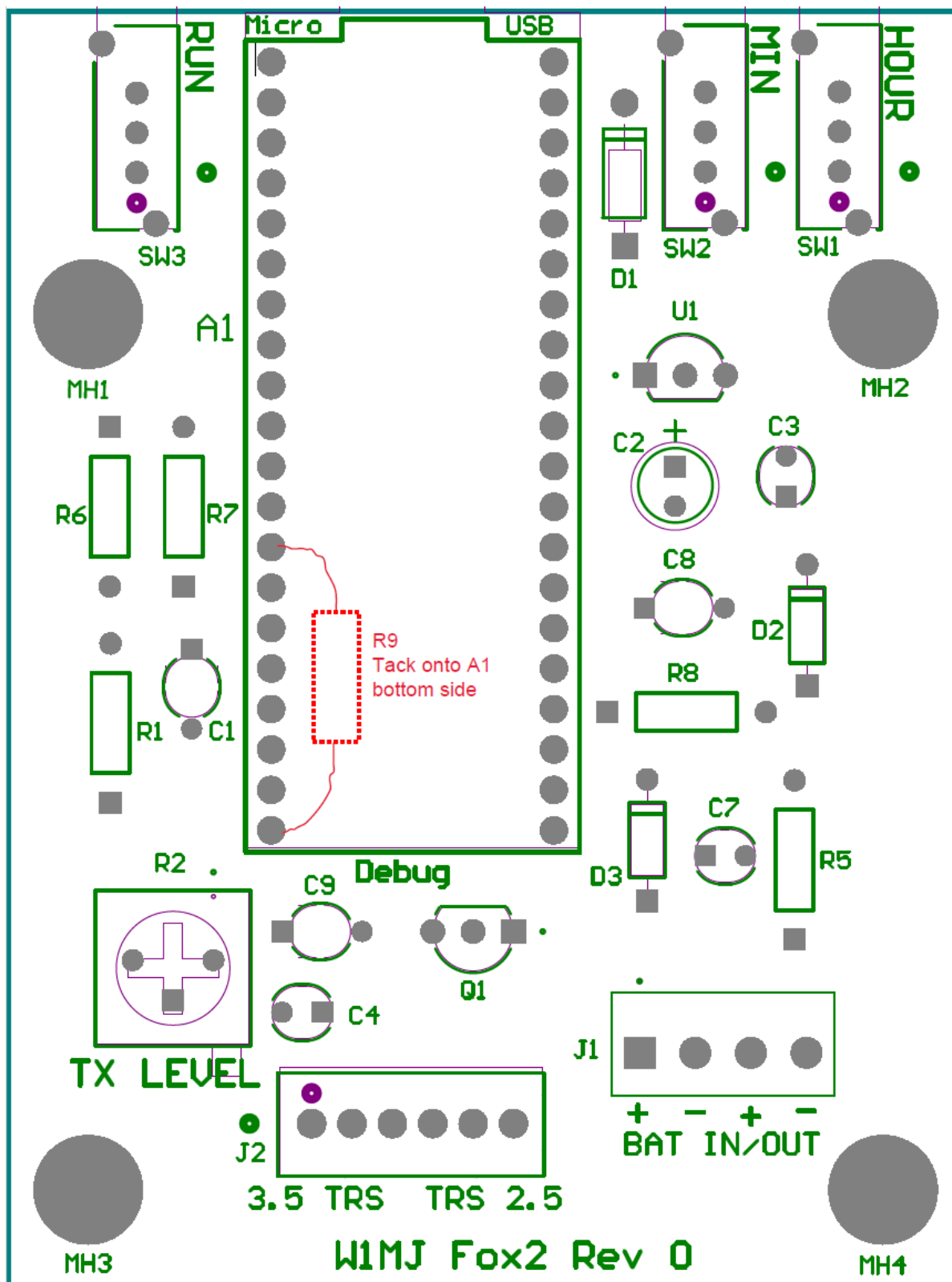


For ease of assembly all components are through-hole. Unfortunately, the board does not have mounting holes for resistor R9; it must be tacked onto the bottom the board as shown in Figure 3. Otherwise, assembly is straightforward.

Table 2 - Bill of Material, Circuit Board

Description	Designator	Qty	Manufacturer	Manufacturer Part #
Raspberry Pi Pico	A1	1	Raspberry Pi	Raspberry Pi Pico
CAP CER 2.2UF 25V X5R RADIAL	C1, C9	2	TDK	FG28X5R1E225KRT06
CAP 2.2UF TANT	C2	1	Kyocera AVX	TAP225K016SRW
CAP CER 0.1 UF 50V X7R RADIAL	C3, C4	2	Kemet	C320C104K5R5TA
CAP CER 0.01 UF 50V X7R RADIAL	C7	1	Kemet	C315C103K5R5TA
CAP CER 10UF 25V X5R RADIAL	C8	1	TDK	FG28X5R1E106MRT06
DIODE SCHOTTKY 20V 1A 1N5817-T	D1	1	Diodes Inc	1N5817-T
DIODE SCHOTTKY 100V 150MA DO35	D2, D3	2	STMicroelectronics	BAT46
TERM BLK 4POS	J1	1	On Shore Technology	EDX555/4DS
TERM BLK 6POS RADIO INTFC	J2	1	On Shore Technology	OSTVN06A150
N-Channel Enhancement-Mode MOSFET	Q1	1	Microchip	TN0702N3-G
RES AX 4.7K 5% 1/4W	R1	1	Yageo	RS125 (Kit)
RES TRIMPOT 1K 0.5W TOP ADJ	R2	1	Bourns	3296P-1-102LF
RES AX 220 5% 1/4W	R5	1	Yageo	RS125 (Kit)
RES AX 4.22K 1% 1/4W	R6	1	Yageo	MFR-25FBF52-4K22
RES AX 1K 1% 1/4W	R7	1	Stackpole	RNMF14FTC1K00
RES AX 47K 5% 1/4W	R8	1	Yageo	RS125 (Kit)
RES AX 150K 5% 1/4W	R9	1	Yageo	RS125 (Kit)
Pushbutton Switch, Right Angle	SW1-SW3	3	C&K	PN11SHNA03QE
Pushbutton, Black	SW1 (Hour)	1	C&K	G004A
Pushbutton, Gray	SW2 (Min)	1	C&K	G004G
Pushbutton, Black	SW3 (Run)	1	C&K	G004R
VREG 5V 250mA L4931CZ50-AP	U1	1	STMicroelectronics	L4931CZ50-AP
PC Board	None	1	Digi-Key (DK Red)	3237-PCB_BARE_BOARD-ND

Figure 3 - Assembly Drawing, Rev 0A



For a photo of the assembled board, see Figure 7 in the Final Assembly section.

Software Installation

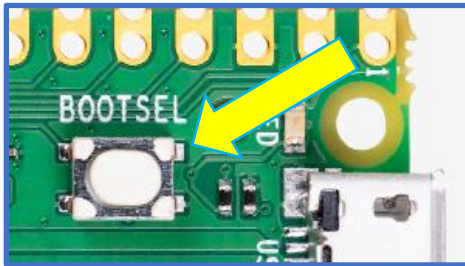
Software may be installed at the end of circuit board assembly. It can be done later, but not if the circuit board is in an enclosure with no access to the BOOTSET button. There are two basic steps that must be performed in order:

1. Install CircuitPython.
2. Install the fox software and audio files.

CircuitPython Installation

The official instructions for installing CircuitPython are found at [here](#). The steps using a Windows PC are:

1. Download the file Raspberry Pi Pico version of CircuitPython from [here](#).
2. Connect a USB cable to the Pico's Micro USB port; don't plug the other end into the PC yet.
3. Press and hold down the BOOTSEL button on the Pico.



4. Connect the USB cable to the PC.
5. Wait a few seconds, then release the BOOTSEL button.
6. Open File Explorer and confirm the presence of drive "RPI-RP2".
7. Using File Explorer, copy the downloaded CircuitPython (.uf2) file to RPI-RP2.
8. Wait a few seconds, then verify that drive RPI-RP2 has been replaced by a new drive named "CIRCUITPY (E:)" or similar. This drive will now be referred to as "the Pico".

Fox Software Installation

Download the fox software, file code.py, from https://github.com/eliotmayer/W1MJ_Fox2, and copy it to the Pico. The Pico drive should now show a file named code.py.

One of the messages in the /talk folder is battery_voltage_is.mp3. You should replace this with a file of the same name that has your own call, e.g. "The K1MJC fox battery voltage is". In other words, don't bootleg my call!

The rotating set of audio messages transmitted by the fox are not considered to be part of the software. See the Message Creation & Installation section.

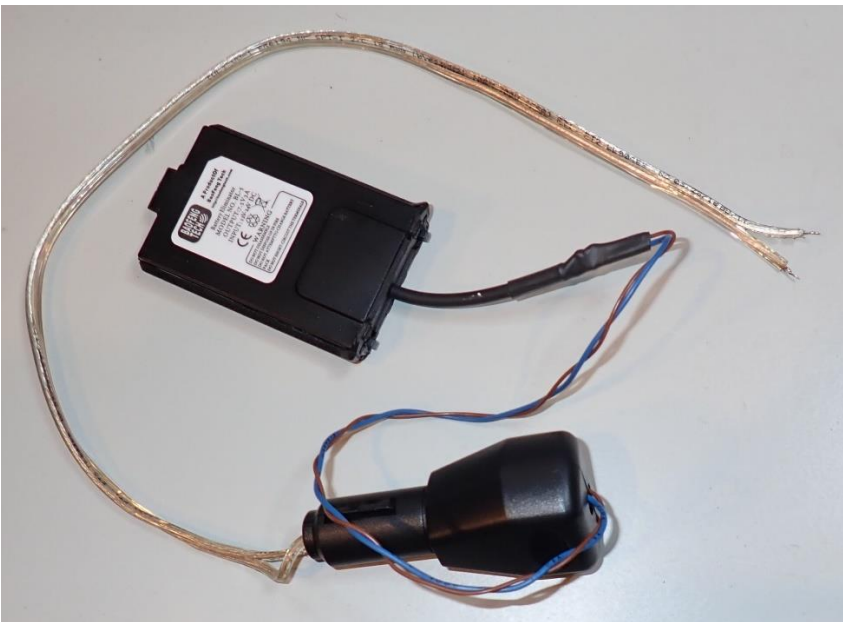
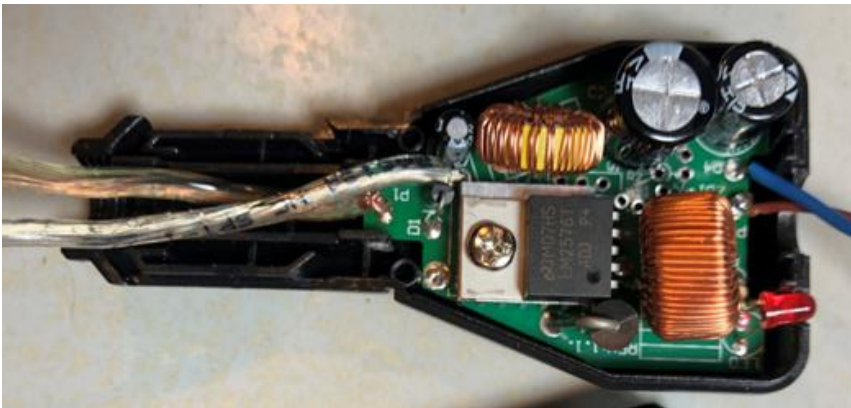
Battery Eliminator Modifications (Optional)

The Battery Eliminator can be used without modifications, as was done in the original W1MJ fox. But there are a few disadvantages:

1. A [car power receptacle](#) (AKA cigarette lighter socket) must be added to the cabling.
2. The car power connectors are not extremely reliable; on one of my deployments, I had to jiggle the connection to make the fox function.
3. The receptacle and the coiled cable take up a lot of space in the fox box.

The alternative is to open up the eliminator's plug, remove the 12V tip and the coil cord, and solder the DC wiring directly to the battery eliminator circuit board, as shown below. Cut the wire from battery case just before it starts to coil. Perhaps that end could be soldered directly to the eliminator circuit board, but I added some extra wire.

Figure 4 - Battery Eliminator Modification



Antenna Assembly

The fox uses a twin lead J-Pole antenna – an excellent suggestion from John Flood, N1JAF. It has Velcro cable strap at the top to make it easy to hang from a tree branch during deployment

The antenna was assembled per this article by WB3GCK: <https://www.qsl.net/wb3gck/jpole.htm>

The feedline is RG58 coax, wrapped 3 times around an Amidon FT140-43 ferrite core to help isolate the feedline from the antenna.

Here is related article about twin lead J-pole antenna:

<https://www.arrl.org/files/file/Public%20Service/TrainingModules/jpole-dual-band.pdf>

Figure 5 - J-Pole Antenna



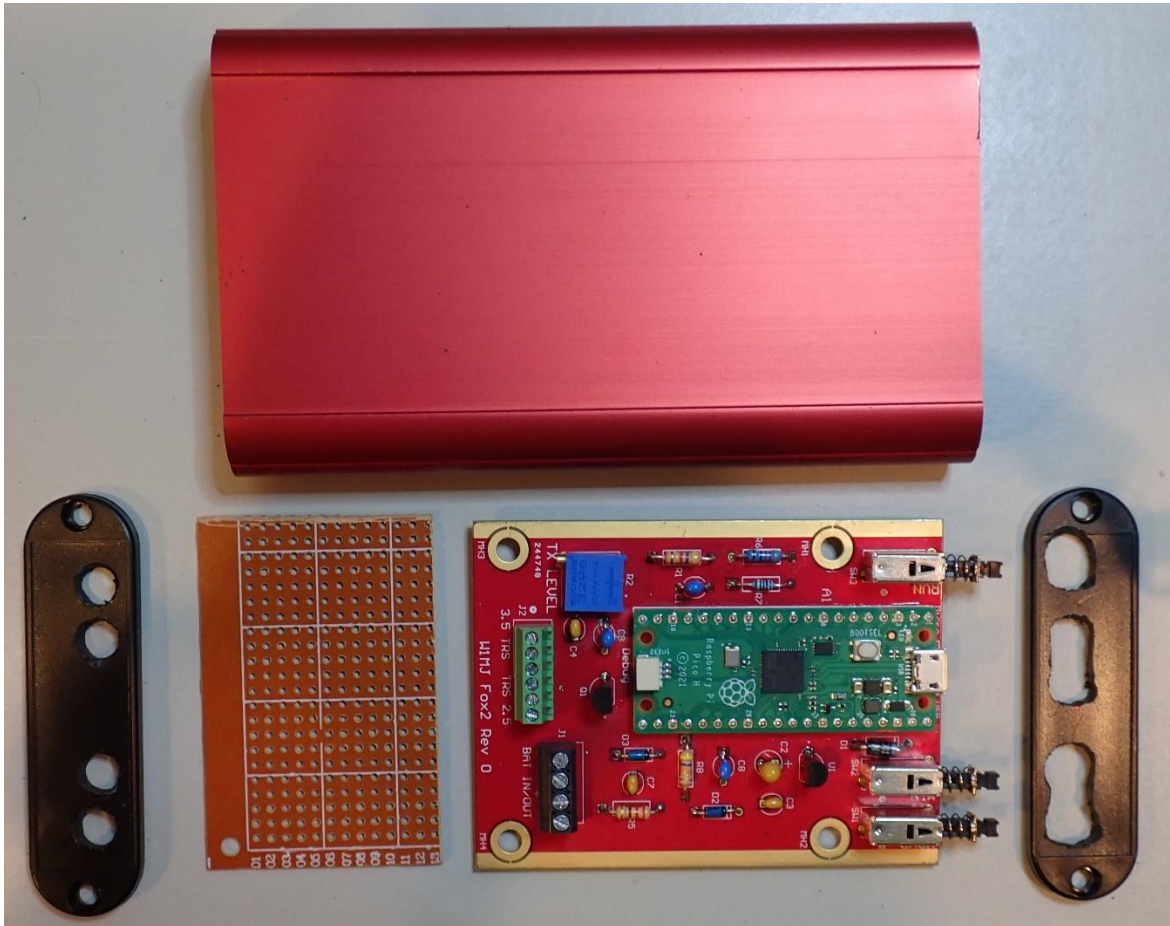
Circuit Board Enclosure (Optional)

Mounting the circuit board in its own box was done primarily for aesthetics. It probably is not necessary, so long as care is taken not to damage the components during final assembly and during deployment. Protection from the elements is already provided by the fox case.

I used a Hammond 1455B1202RD case. The PC board width was chosen to fit this case's integral card guides. The board length was not extended to fit the box, as that would have (1) raised the PC board fabrication cost and (2) taken up extra space for anyone wishing to use this board in an alternative enclosure. The extra length of the card guide is instead filled with a scored-off piece of inexpensive prototyping board.

The PC board has 4 mounting holes that can accommodate 6-32 screws for use in an alternative enclosure.

Figure 6 - Circuit Board Enclosure



Final Assembly

Connect the components per Figure 1 and Figure 7, noting that:

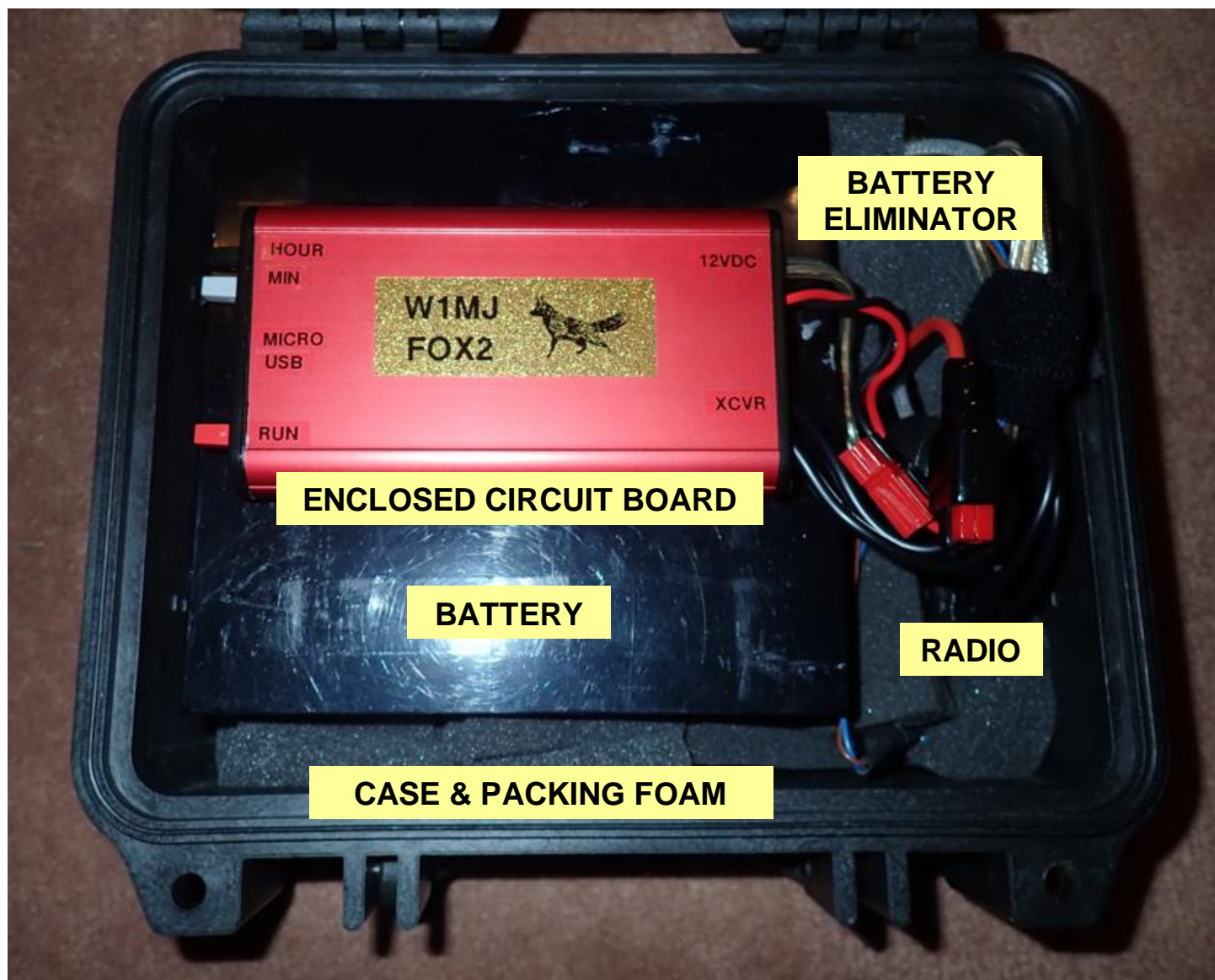
- Terminal block J1 has connections for 12VDC input from the battery and output to the radio's battery eliminator (Figure 10)
- Not all J2 connections to the 2.5mm and 3.5mm radio jacks are used (Figure 10), but hooking them up provides some extra mechanical support. The "TRS" in the silkscreen indicates the Tip, Ring, and Sleeve connection for each cable.

Figure 7 - Assembled Circuit Board with Cables Connected



All components fit into the Pelican case, and the foam pieces that come with case are used as needed to keep the components from shifting around during transport.

Figure 8 - Final Assembly



Message Creation & Installation

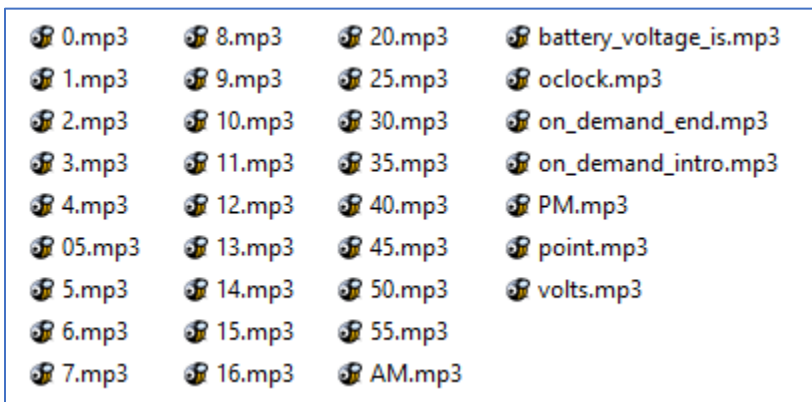
The fox messages must be stored as .mp3 files in the /messages folder. They will be played in alphabetical order, so name them accordingly. After the last stored message is transmitted, the battery voltage will be announced, and then the messages will repeat.

Required .mp3 files for announcing time and voltage are stored separately, in the /talk folder. Most message say the same as the file name, e.g. 45.mp3 says, “forty five”. Exceptions:

battery_voltage_is: “The W1MJ fox battery voltage is “.

on_demand_intro.mp3: “The W1MJ fox has received a request and will be on the air until ”.

on_demand_end.mp3: “This is the last fox transmission. Make a voice request for more. W1MJ”



Hi-Fi audio files are not required, which is a good thing given the limited storage capacity of the fox memory. The following MP3 file parameters are recommended:

- Sample Rate: 16 kHz
- Channels: Mono

With these settings, the file size of 20-second messages is approximately 40 kb. Storage is available for at least 20 such messages

Even an 8 kHz sample rate produces acceptable files, and they are half the size of the 16 kHz files. This sampling rate is appropriate if one wants to transmit many different messages before they repeat. I *think* the audio quality was slightly lower than with 16 kHz sampling, but the difference, if any, was small. Figure 9 shows an example of MP3 file rendering for minimum file size using [Reaper DAW](#).

Figure 9 - MP3 File Rendering with Reaper DAW

Render to File

Source: **Master mix** Bounds: **Time selection** Presets

Time bounds
 Start: **10:55.363** End: **11:13.423** Length: **0:18.060** ☐ Tail: **1000** ms

Output
 Directory: **T:\Eliot\Music\Reaper Projects\Fox** Browse...
 File name: **small_file_test_3** Wildcards
 Render to: **T:\Eliot\Music\Reaper Projects\Fox\small_file_test_3.mp3** 1 file

Options
 Sample rate: **8000** Hz Channels: **Mono** Full-speed Offline
☒ Use project sample rate for mixing and FX processing ☐ 2nd pass render
 Resample mode: **Sinc Interpolation: 192pt** Normalize ON
☐ Tracks with only mono media to mono file ☐ Dither master ☐ Dither stems
☐ Multichannel tracks to multichannel files ☐ Noise shape master ☐ Noise shape stems
☐ Only render channels that are sent to parent ☐ Render stems pre-fader
 Metadata... Embed: ☐ Metadata ☐ Stretch markers/transient guides ☐ Take markers

Primary output format **Secondary output format**

Format: **MP3 (encoder by LAME project)**
 Mode: **Constant bitrate (CBR)** Better q=2 (recommended)
 Quality: **10 (worst)** approximate ☐ Do not allow joint stereo
 Bitrate: **8 kbps** ☐ Write ReplayGain tag
 LAME 3.99.5

☐ Silently increment filenames to avoid overwriting ☐ Add rendered items to new tracks in project
☐ Do not render files that are likely silent ☐ Save copy of project to outfile.wav.RPP
☐ Save outfile.render_stats.html

Queued Renders... Dry Run (no output) **Render 1 file**
 Cancel Save Settings

Configuration

Prior to deployment, several configuration settings can be modified by connecting the fox to a computer USB port. The default values are shown in brackets:

- Message Interval (time between start of transmissions) [60 seconds]
- Activation time for Scheduled Mode (daily) [8:00 AM to 8:00 PM]
- On-Demand Mode settings:
 - Run time per request [60 minutes]
 - Minimum RX level [0.25 volts] (for BaoFeng UV-5R with volume control at maximum)

- Minimum duration of request [2.0 seconds]
- Minimum battery voltage [12.2 VDC, about 10% charge for LiFePO4 batteries]
- Battery voltage correction factor (to make announced voltage agree with DMM reading) [0.985; if calibration is not of concern, change this to 1.0]

The fox appears as an external drive, and the Settings section is near the top of file “code.py”. Any text editor can be used.

The messages to be transmitted should be configured prior to the initial deployment. See Message Creation.

The transceiver should be configured to the desired frequency and power level. Its volume control should be set to maximum, and the squelch should be set just high enough to prevent opening accidentally.

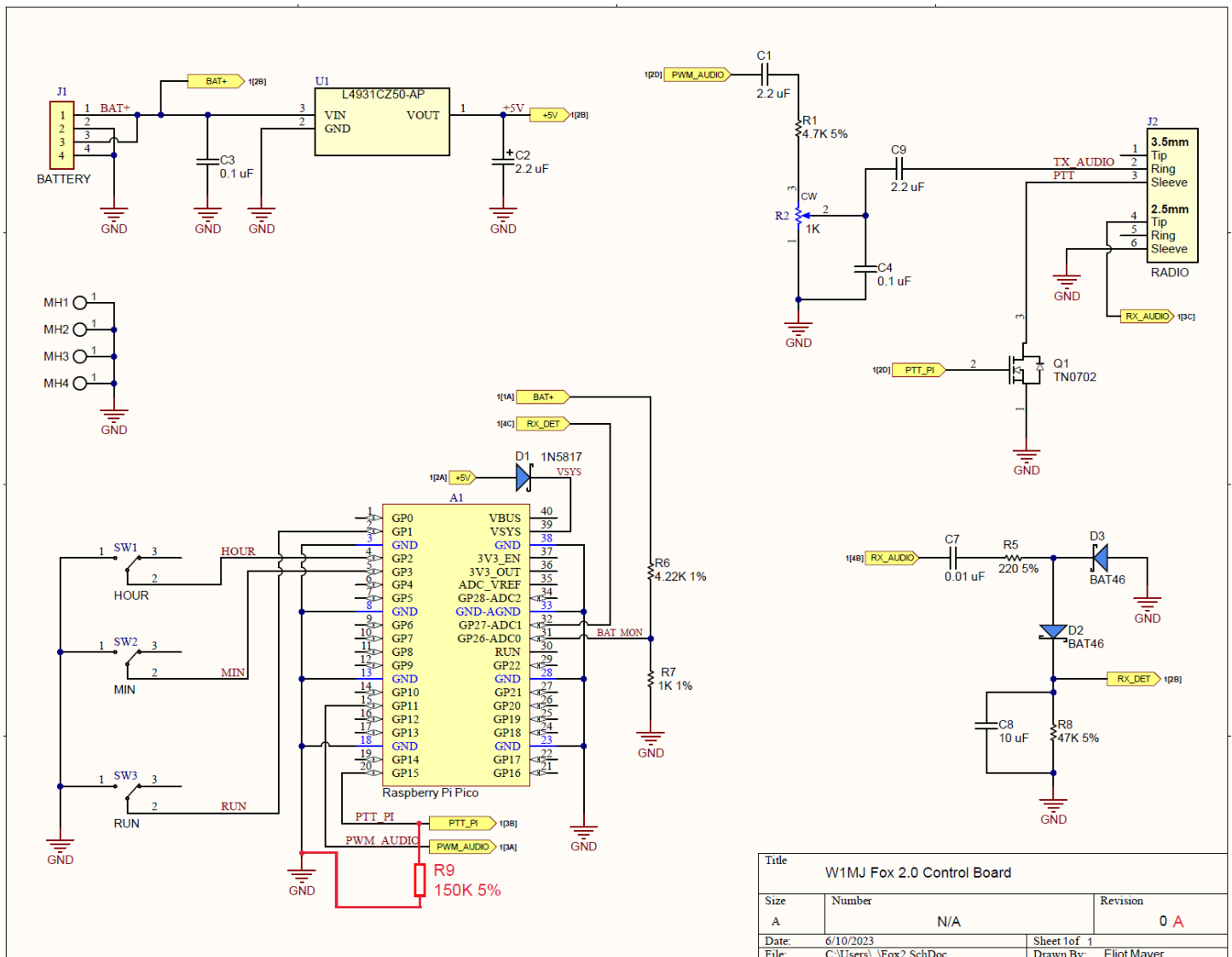
The control board has a trimpot for transmitted audio level. It should only be necessary to set this once, by ear, for normal audio level.

Deployment

Go to the hiding location, which is typically a public location such as conservation land. Bring a separate radio along.

1. Prior to connecting the battery to the fox at the hiding location, tune the separate radio to the fox frequency.
2. If you have not done so before deployment, set the fox radio to the desired frequency, and set its volume control to maximum.
3. Power up the fox. You should hear the battery voltage announcement.
4. Set the time of day to the nearest 5 minutes using the HOUR and MIN buttons. Each time update is sent over the air.
5. Start operation in the desired mode, as follows:
 - Scheduled Mode: Press the RUN button briefly (less than 1 second). Message transmission will begin if the current time is within the schedule.
 - Start On-Demand Mode: Press and hold the RUN button for more than 1 second. To verify operation, make a transmission for 2 seconds or longer on the other radio. The fox should announce that it has received a request.

Figure 10 - Schematic Diagram, Fox Circuit Board



Pico Power Supply

I considered using a switching power supply to provide +5V to the Pico. However, the Pico's current consumption is fairly low, so it didn't seem worth the complication. A simple linear regulator was used, U1.

Transmit Audio Interface

The high frequency component of the PWM audio is filtered out primarily by R1 and C4. Trimpot R2 sets the transmit audio level. Capacitors C1 and C9 block DC components of the PWM audio and the radio mike input, respectively.

PTT Circuit

MOSFET Q1 keys the transmitter PTT when the PTT_PI signal is set high.

From the “Don’t believe everything you read online” department: I read that the Pico IO ports default to inputs with weak pull-downs. That should prevent any accidental PTT keying. But it turns out that if no program is running, or if the program stops running for any reason, PTT_PI goes into a high impedance condition. That intermittently results in random PTT keying. This oops was solved by adding pull-down R9.

Receive Audio Detection Circuit

Received audio is rectified by diode D2. The positive peaks are stored on C8, which is slowly discharged by R8. The resulting RX_DET signal is measured by one of the Pico’s analog inputs when the fox is waiting for requests in the On Demand mode.

Battery Monitor

The Pico’s analog inputs measure the range 0 to +3.3 VDC. Therefore, the battery voltage is reduced by voltage divider R6 / R7 for measurement. The software scales the measurement accordingly, and even has a calibration value that can be set to compensate for gain errors caused by the resistors and the Pico’s circuitry. To keep the measurements as accurate over temperature in the field, R6 and R7 were selected for low-TCR (temperature coefficient of resistance).

PC Board

The circuit is simple enough that it could be assembled onto a prototyping board. However, I received training at work in Altium Designer, and decided to reinforce the training by making a PC board for this circuit.

Fabrication cost for a small quantity of boards is reasonable these days. As with other things, the lowest prices are from China. However, since this was my first-ever computer-designed PCB, I went with Digi-Key’s DK Red service to minimize the chance of a communication error if something went wrong. I kept the design rules very conservative so that any future fabrications could be done just about anywhere.

Schematic symbols and PCB footprints are available online for many components. For example, I Googled, “altium symbol for raspberry pi pico”, and found <https://github.com/amgsus/RPi-Pico-Altium>, though I tweaked it a bit to my liking. Many of the symbols could be found right from Digi-Key web page for the components that I ordered.

My design rule notes:

[DKRed Design Rules](#)

Cheap fab found via Google: <https://www.pcbway.com/orderonline.aspx>

Another (from a negative review of PCBWay): <https://jlcpcb.com/>
(but cannot find all design rules, such as line width)

Parameter	DKRed Spec	PCBWay Spec	W1MJ Fox Design Rules
Minimum Trace Spacing	5 mils (.13 mm)	6 mil	8 mils (0.20 mm)
Minimum Trace Width	5 mils (.13 mm)	6 mil	8 mils (0.20 mm)
Hole Plating Thickness	1 mil (.025 mm)		1 mil (0.03 mm)
Minimum Drill Size	8 mils (.20 mm)		10 mils (0.25 mm)
Minimum Via Hole Size	8 mils (.20 mm)	0.3 mm	12 mils (0.30 mm)
Minimum Via Pad Size	16 mils (.41 mm)		20 mils (0.51 mm)
Minimum Pad Size	10 mils (.25 mm)		n/a (no SMT parts)
Accuracy Tolerance	5 mils (.13 mm)		n/a

During layout, I re-assigned the Pico IO pins as needed for easy routing. This allowed for all routing except for GND to be done on the bottom layer. The top layer is a dedicated ground plane.

For ease of assembly:

- All components are through-hole
- Connections to the ground plane have above average thermal relief – fewer, longer, skinnier spokes than the Altium Designer default value.

Note to self: Altium Designer project is on my work laptop in folder Documents\Misc\W1MJ\Fox2.

Software Design Overview

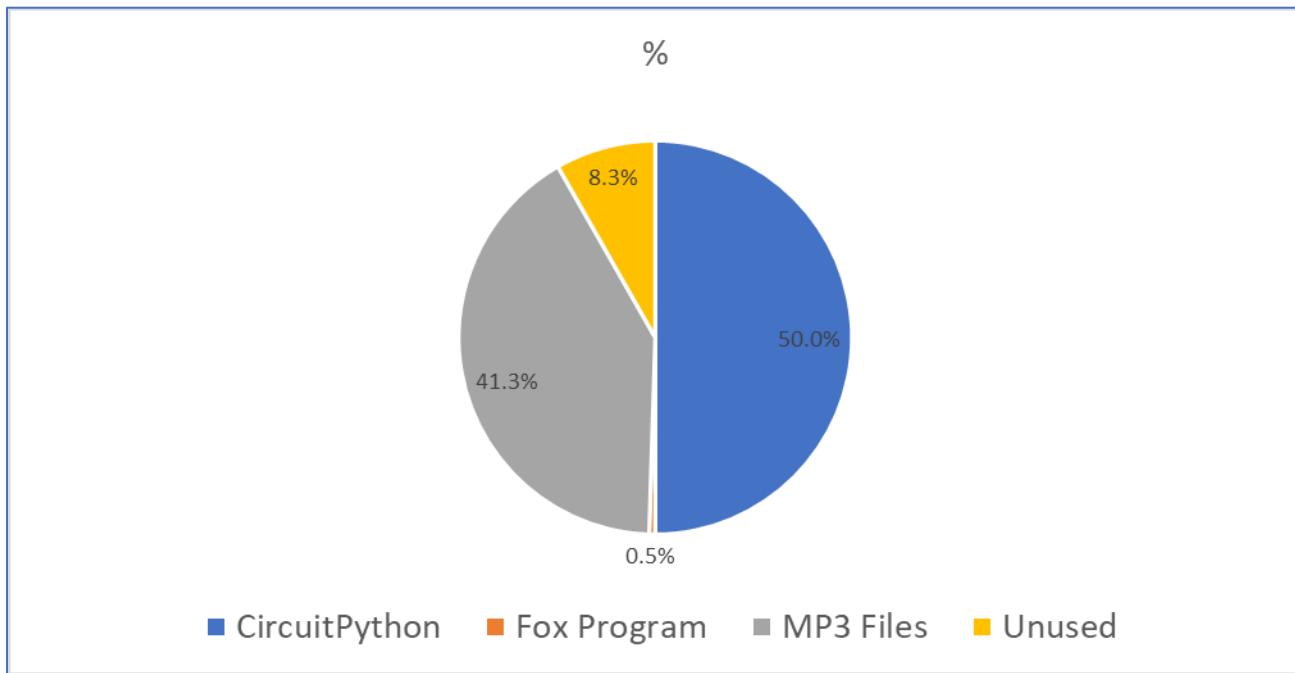
Choice of Programming Language

Python is a popular high-level language. It is interpreted rather than compiled, which means that one can try a new program or bug fix with no extra compiling steps. Even single lines of code can be run in order to master a new command. Decades ago, interpreted languages were considered too slow for serious programming, but this is seldom the case given today's processor speeds.

MicroPython is a version of Python suitable for microcomputers. CircuitPython is a variation of MicroPython that includes support for MP3 file decoding. The small file size of MP3 files allows for a lot of messages in the small storage space of the Pico (20 messages). Creating an MP3 decoder from scratch looks like a daunting task to me, so CircuitPython was the best choice for the fox.

Pico Storage Usage

Using CircuitPython and MP3 files, the 2 MB of the Pico is utilized as follows:



Use of Artificial Intelligence

Beware - AI could steer you in the right direction or in the wrong direction! Using openai.com, I got very different results depending upon what I asked. The answer to Question 1 was discouraging, and wrong. The answer to Question 2 was fairly good, providing code similar to what I had written, though its choice of the `audioio` module was poor because it is not available on the Pico. Fortunately, similar functionality is available for the Pico in the `audiopwmio` module.

Question 1: How can I make a Raspberry Pi Pico play MP3 files?

Answer 1: The Raspberry Pi Pico, being a microcontroller board, doesn't have the necessary hardware and processing power to directly play MP3 files like a computer would..."

Question 2: Write a circuitpython program to decode and mp3 file and play it using the RP2040 PWM output.

Answer 2:

Here's an example program in CircuitPython to decode and play an MP3 file using the RP2040 PWM output:

```
python Copy code

import board
import digitalio
import audiomp3
import audiocore
import audioio

# Initialize the PWM output pin
pwm = digitalio.DigitalInOut(board.GP2)
pwm.direction = digitalio.Direction.OUTPUT

# Open the MP3 file
with open("test.mp3", "rb") as f:
    mp3 = audiomp3.MP3Decoder(f)

# Create an AudioOut object and connect it to the PWM output
audio = audioio.AudioOut(pwm)
audio.play(audiocore.DecodedSample(mp3))

# Wait for the playback to finish
while audio.playing:
    pass

# Stop the PWM output
audio.stop()
pwm.value = False
```

This program uses the RP2040's onboard PWM output and the Adafruit audiomp3 library to decode and play an MP3 file. The program assumes that the MP3 file is named "test.mp3" and is located in the same directory as the Python script.

The program first initializes the PWM output pin and opens the MP3 file. It then creates an AudioOut object and connects it to the PWM output, and starts playing the decoded audio data using the audiocore.DecodedSample class.

The program then waits for the playback to finish by checking the AudioOut object's "playing" property in a loop. Once the playback has finished, the program stops the PWM output and sets the output pin to a low state.

Answer 2 was much better. The audiomp3 part is spot on. Unfortunately, the audioio module is not supported for the Pico, but the audiopwmio module serves the same purpose.

CircuitPython Modules

The CircuitPython modules are detailed here: <https://docs.circuitpython.org/en/latest/shared-bindings>

Each module has an expandable "Available on these boards" list, and not all modules are available on the Pico.

Program Outline

1. Import modules
2. Settings (message interval, start & stop times, etc)
3. Define Functions
4. Initialization
 - a. IO Port assignments
 - b. Miscellaneous initialization tasks
5. Set time (buttons, voice feedback via radio)

Buttons: Hour, Min (5-min increments), Run (long push for On Demand Mode)
5. Message Loop
 - a. During scheduled time, with OK battery voltage:
 - i. Transmit all messages sequentially
 - ii. Measure and transmit battery voltage
 - iii. Repeat
 - b. In On Demand Mode, wait for new request

Software Design Details

Time

Ref: time module, <https://docs.circuitpython.org/en/latest/shared-bindings/time/index.html>

It wasn't clear from the available information, but the Pico behaves as if has a fairly accurate real-time clock. At power up, it thinks that it is the start of the year 2020. The fox uses pushbuttons to allow setting of the correct time of day to the nearest 5 minutes; it ignores the date. It uses these functions from the time module:

- `time.time()` returns the number of seconds since the beginning of 1970.
- `time.localtime(t)` converts the above time into a structure (year, month, day, etc.)
- `time.sleep(t)` waits for the specified time in seconds.

The fox adds setting `t_power_up_hr_min [hour, min]` to the initial time of day (midnight at power up). With the default value of [8,0], and the fox just powered up, the time setting begins at 8:00 AM instead of 12:00 AM.

Key time variables:

- `t_now`: Current time in seconds, from `time.time()` with correction per the time setting buttons. It is limited to 5 minute increments in order to limit the number of mp3 files needed to announce the time.
- `t_correction`: The correction factor to be added to `time.time()` to create `t_now` at various points in the program.

In order to easily keep track of the daily start and stop times for transmissions, the program includes function `time_of_day_mins(t)` to convert time to the time of day in minutes. It recognizes the input `t` as either an integer (such as `t_now`) or an [hour, min] list (such as `t_start_hr_min`).

MP3 Decoding

CircuitPython's `audiomp3` module makes mp3 decoding easy. For details, see <https://learn.adafruit.com/mp3-playback-rp2040>, but ignore the hardware section and scroll down to the software part.

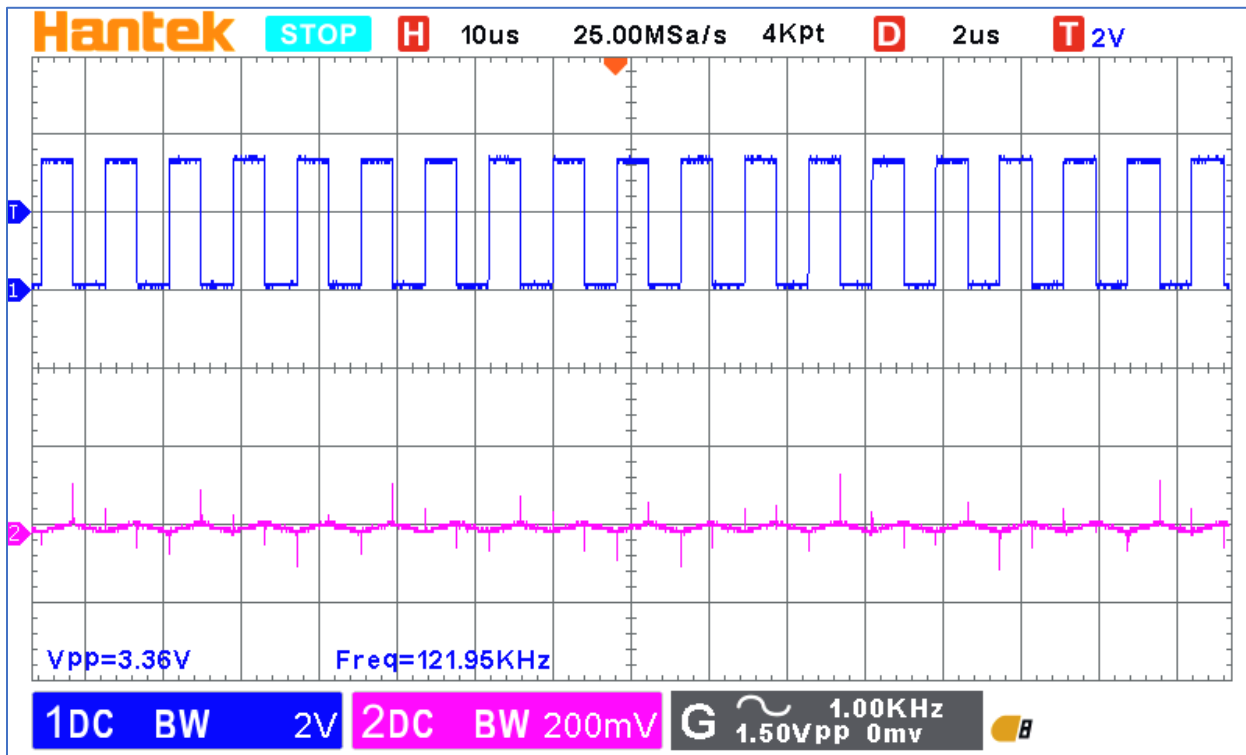
Unfortunately, there is a noticeable glitch at the end of the message due to the sudden shift in average DC level from 1.65V to 0. The IO port goes into a high-impedance state, so I tried biasing the output to 1.65V with a voltage divider (3.3V – 180K – 180K – GND). That didn't help because the MP3 function brings the port to 0V for a brief time before it goes to high impedance. The glitch is somewhat annoying, but livable. A potential option is to time the PTT cleverly against the beginning (easy) and end (not so easy) of the mp3 files.

PWM Output

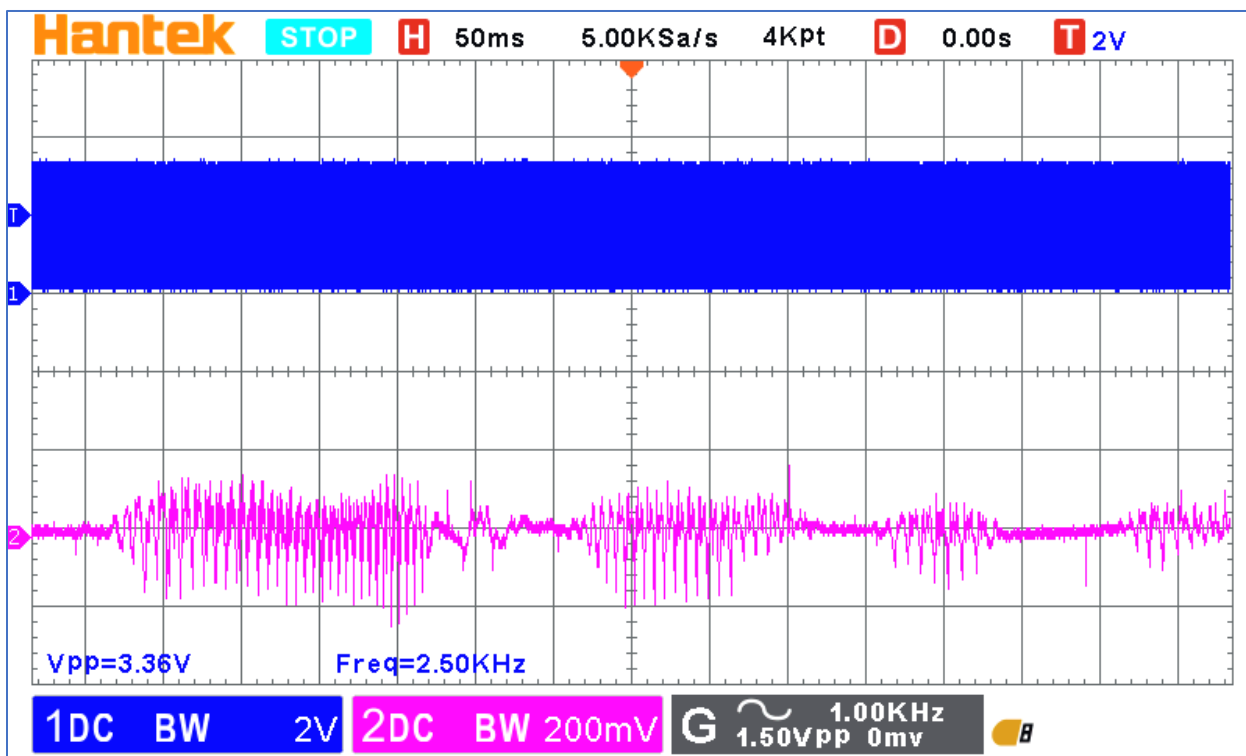
The decoded mp3 audio files are output in PWM (pulse-width modulation) format using CircuitPython's `audiopwmio` module. There is apparently no control over the PWM parameters, but this is not a problem. During development, I observe a fixed-frequency, pulse-width modulated output, running at approximately 122 kHz .

Scope traces:

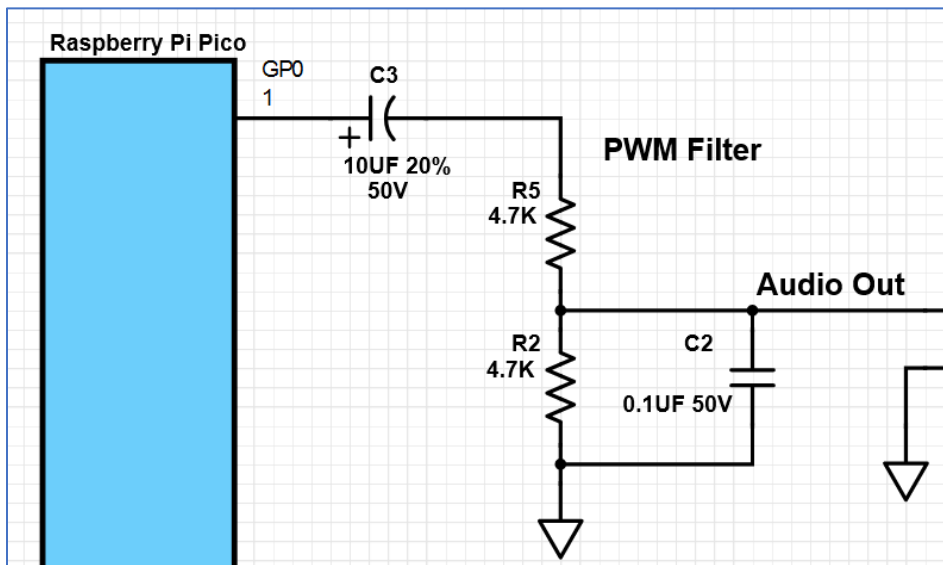
1. IO Port GP0: Fixed-frequency, pulse-width modulated output, running at approximately 122 kHz .
2. Audio output using the passive filter circuit shown below.



As above, zoomed out in time to show voice waveform on Trace 2. With this simple passive filter, there is still a small 122 kHz component, which will hopefully be ignored by the transmitter.



The following PWM filter was used for the scope photos. For audible tests, lower resistor R2 is 470Ω to provide a level suitable for listening with a Honeytone guitar amp. The final filter design is shown in the Fox Circuit Board Design section.



Analog Input

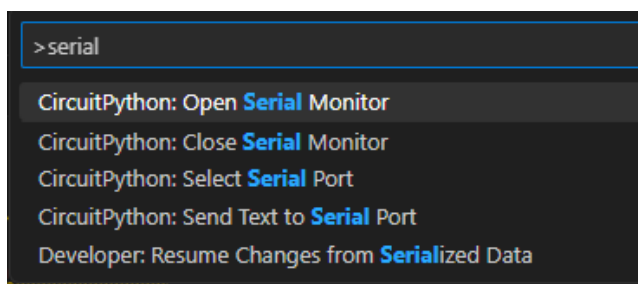
CircuitPython's analogio module makes it easy to read the battery voltage and receiver audio levels. See <https://learn.adafruit.com/circuitpython-essentials/circuitpython-analog-in>.

Software Development Tools

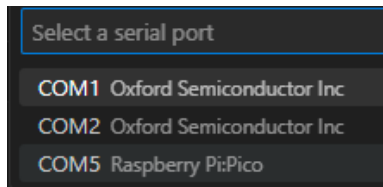
The Mu Editor, recommended by Adafruit, works well.

Visual Studio Code works with the CircuitPython extension but is more complicated to use. Unlike the Mu Editor, the VS Code Terminal window does not automatically figure out the COM port. One has to do this:

- Start the command palette (Ctrl + Shift + P)
- Type “serial” then click CircuitPython: Select Serial Port



- Select the Raspberry Pi Pico



- From the “serial” list (above) it may also be necessary to click “CircuitPython: Open Serial Monitor”.

When all is set, the Terminal window behaves like the Serial window in Mu Editor.

```
Circuit Python Serial Monitor
[Open] Connection to COM5

press Ctrl-C to enter the REPL

soft reboot

Auto-reload is on. Simply save files over USB to run them or enter REPL to disable.
code.py output:
There are 3 files in the message folder.

Code done running.

Press any key to enter the REPL. Use CTRL-D to reload.
█
```

ⁱ https://en.wikipedia.org/wiki/Transmitter_hunting