Python Radio 32: Old-Time Radio

Build a Time Machine

Simon Quellen Field

Simon Quellen Field

Follow

6 min read

·

Jan 5, 2025

Listen

Share

More

Press enter or click to view image in full size

Old-Time Radio Time Machine

MidJourney

In the days before television, electronic entertainment meant radio.

Not just music and news, but comedy shows, mysteries, and adventure came streaming into homes through receivers that were often furniture. Fancy polished wood cabinets were centerpieces in the living room, with chairs arranged around them so the family could listen together.

Those were the days of Abbott and Costello, Jack Benny, Burns and Allen, Amos and Andy, and Fibber McGee and Molly. There was the CBS Radio Mystery Theater. There were dramas like Our Miss Brooks and Sherlock Holmes.

Thousands of these old radio shows are now collected on the Internet to be downloaded for free, as they are now out of copyright. Orson Welles’ War of the Worlds caused a panic when people believed it was a true news program. The Lone Ranger brought the old West into living rooms.

Many of the older shows from the 1930s are poorly recorded, but by the 1940s the technology had improved greatly, and the sound quality is quite good.

My Receiver

I have a replica of an old-time radio. It has a nice polished wood cabinet and sits on the counter in the living room. Under it is a drawer where I have hidden an A.M. radio transmitter playing a mix of those old-time radio shows.

Visitors find this quite amusing.

My Transmitter

In keeping with the theme of this series, the transmitter is an ESP32 Lolin S2 Mini connected to an SD card reader and a PCM5102 stereo Digital to Analog Converter (DAC) board.

The ESP32 sends out a 3.3-volt 625-kilohertz square wave that acts as the carrier. A diode that amplitude modulates the carrier mixes it with the audio from the DAC.

Press enter or click to view image in full size

The Transmitter.

Photo by the author

The connections for the SD Card Reader are as follows:

ESP32 Pin 5 → SD Card Reader CS

ESP32 Pin 18 → SD Card Reader SCK

ESP32 Pin 12 → SD Card Reader MOSI

ESP32 Pin 11 → SD Card Reader MISO

ESP32 Pin 3.3V → SD Card Reader VCC

ESP32 Pin GND → SD Card Reader GND

Press enter or click to view image in full size

The SD Card Reader

Photo by the author

The connections for the PCM51002 DAC are:

ESP32 Pin 40 → DAC BCK

ESP32 Pin 38 → DAC DIN

ESP32 Pin 36 → DAC LRCK

ESP32 Pin 3.3V → DAC VIN

ESP32 Pin GND → DAC GND

Press enter or click to view image in full size

Photo by the aurhor

Since the ESP32 only brings out one pin for 3.3 volts, I made a Y connector to power both of the other boards.

The PWM output for the carrier wave comes out on pin 3. This goes to a 1k ohm resistor. The audio output from the DAC comes out from the left output pin (marked L on one side of the board and LROUT on the other). This goes to another 1k ohm resistor.

The two resistors connect to one end of the diode (any silicon diode will work).

Press enter or click to view image in full size

The A.M. modulator

Photo by the author

The remaining end of the diode connects to a loop of wire whose other end connects to the analog ground pin on the DAC. This loop of wire is our antenna. You can make it as long as you like.

How the Hardware Works

We have two signal sources — the 625-kilohertz carrier and the audio signal, which varies between perhaps 100 hertz up to at most about 20 kilohertz.

To get an amplitude-modulated signal from these, we need to multiply them together. You might think we would use the computer to do that, but instead, we use a five-cent diode.

How can a diode do multiplication?

When we apply a voltage to a diode, the current going through the diode rises exponentially with the applied voltage. The graph looks like this:

Press enter or click to view image in full size

Voltage/current graph for a diode

Mmerevise.co.uk

When we add exponents, we get multiplication. We put our two signals in at one end, and their voltages add in the diode. The result is the product of the two. Since the signals are made up of waves, they interfere, and we get our amplitude-modulated signal out.

What, No Filter?

Our carrier wave is a square wave. It consists of the fundamental sine wave, plus all of the odd harmonics of that wave, in decreasing amplitude as they increase.

This means we could hear our old-time radio at 1875 kilohertz, but only one-third as far. We could hear the signal at 3125 kilohertz, but only 1/5th as far.

We could build a low-pass filter to remove these harmonics. But our signal is 3.3 volts, at about 10 milliamperes, into an antenna that is ten inches long. The wavelength at 625 kilohertz is over 1,500 feet long so that antenna is far from resonant.

How far does our signal go?

The best I’ve seen is about a foot, after careful fiddling with the position of the antenna and the receiver.

The receiver picks up a beautifully clear signal when it is a few inches above the hidden transmitter. That signal and clarity would be the envy of any 1940s radio listener. But move the receiver a few inches, and the signal degrades quickly.

The Software

Our SD card reader can’t handle the 64-gigabyte microSD cards I first tried. It could probably handle 32-gigabyte cards, but I found a one-gigabyte card in a drawer that I’m sure no one will miss. It will hold over 300 radio shows of about ten minutes each. That’s two days of constant programming.

We will play WAV files since micropython has modules to do that. Many of the radio programs are in MP3 format, so we convert them to WAV before storing them on the microSD card.

Because the language has support for both boards, the program is actually fairly short:

From machine import I2S, SPI, Pin, PWM

From micropython import schedule

From os import mount, umount

Spi = SPI(2, baudrate=100000, polarity=0, phase=0, sck=18, mosi=12, miso=11)

From sdcard import SDCard

Sdcard = SDCard(spi, cs=Pin(5))

Mount(sdcard, “/sd”)

Pin = Pin(3, Pin.OUT)

Pwm = PWM(pin, freq=10, duty=512)

SCK\_PIN = 40

SD\_PIN = 38

WS\_PIN = 36

I2S\_ID = 0

BUFFER\_LENGTH\_IN\_BYTES = 40000

Wav = open(“/sd/{}”.format(“WhosOnFirstAudio.Wav”), “rb”)

Audio\_out = I2S(I2S\_ID,

Sck=Pin(SCK\_PIN), ws=Pin(WS\_PIN), sd=Pin(SD\_PIN),

Mode=I2S.TX, bits=16, format=I2S.MONO, rate=16000, ibuf=40000)

Silence = bytearray(1000)

Wav\_samples = bytearray(10000)

Wav\_samples\_mv = memoryview(wav\_samples)

Def i2s\_callback(arg):

Num\_read = wav.readinto(wav\_samples\_mv)

If num\_read == 0:

Pos = wav.seek(44)

Audio\_out.write(silence)

Else:

Audio\_out.write(wav\_samples\_mv[:num\_read])

Def freqs():

Print()

Print(“Convenient reachable frequencies in the A.M. band:”)

Guess = 0

For f in range(540, 1700):

Try:

Pwm.freq(f \* 1000)

Actual = pwm.freq()

If actual != guess and actual % 1000 == 0:

Print(str(actual / 1000) + “ kHz”)

Guess = actual

Except ValueError as verr:

Pass

Print()

Def main():

Pwm.freq(625 \* 1000)

Audio\_out.irq(i2s\_callback)

Wav.seek(44) # advance to first byte of Data section in WAV file

Audio\_out.write(silence)

Freqs()

Main()

It would be even shorter, but I threw in a little routine to calculate convenient frequencies in the A.M. band that the ESP32 can reach using its PWM module.

The SD card reader talks to the ESP32 over the SPI bus. We mount it as a file system called /sd.

The PCM51002 DAC talks to the ESP32 over I2S, a streaming protocol we have not used before in this series. We read samples from the WAV file and stream them to the DAC using I2S. The hardware can thus do all of the heavy lifting, and our program is simple.

We start out by sending a buffer full of zeros for a moment of silence. This “primes the pump”, and we get an interrupt when the I2S module wants more data. That’s when we send the WAV bytes.

At the end of the file, we send more silence, rewind to the beginning, and start over.

In this program, I just play Abbott and Costello’s Who’s On First comedy routine over and over. But you can concatenate days’ worth of old-time radio shows into one big WAV file if you like. Or you can read the files off the microSD card individually.

Programming

Python

Radio