Python Radio 8: Hacking a Receiver

Simon Quellen Field

Simon Quellen Field

Follow

6 min read

·

Aug 24, 2024

Listen

Share

More

The ESP32 as a local oscillator

Press enter or click to view image in full size

Photo by author

The little XY-MK-5V receiver we have been using (see photo above) is a type known as a regenerative receiver. This is a very old design, invented in 1912 by an undergraduate at Columbia University named Edwin Armstrong.

Some things to notice: the tuning is done with a tiny screwdriver that moves a ferrite slug in the coil in the center. There is one integrated circuit, the LM358 audio amplifier chip, in the upper right. There are two transistors (the 2SC3356s, marked R25), and a number of surface mount resistors and capacitors, in addition to two more coils, the three-turn coil and the big epoxy-coated coil near the center.

There is a lot going on there, out in the open where we can see it.

But there is a better way to build a radio receiver. It was also invented by Edwin Armstrong (in 1917), and separately by the Frenchman Lucien Levy. This radio mixes the incoming radio waves with a locally generated signal to produce “beat frequencies”, which are the sum and the difference frequencies between the original signal and the local oscillator signal.

This has many advantages. The amplifier stages now only have to deal with one frequency (called the intermediate frequency). Tuning is done by changing the local oscillator.

One such superheterodyne radio receiver designed to replace the XY-MK-5V is the RXB12.

Press enter or click to view image in full size

Photo by author

Right away we can see it looks more modern. Almost all of the electronics are in the SYN470R integrated circuit in the center of the board. There are eight surface-mount capacitors around the chip, and one zero-ohm resistor (used as a jumper — more about that later). But there is also a big metal can in the upper right corner (big is relative, the whole board is only 30 millimeters across). That can holds a quartz crystal which sets the frequency to 433.98 megahertz.

The new receiver has more precise tuning (much like a quartz watch keeps better time than one with moving parts), and it is a little more sensitive, giving better range. I got mine for 58 cents from AliExpress, and you can get one with the transmitter for another 12 cents, about the same as what I paid for the XY-MK-5V and the FS1000a.

Get a few of them, since we are going to butcher at least one as we modify it to allow us to tune it to a huge number of frequencies.

The chip uses the crystal to control its local oscillator. But the chip also allows us to connect our own oscillator instead. Our oscillator is inside the ESP32, in the form of its Pulse Width Modulation feature. The ESP32 can change the state of a pin as fast as 40 million times per second using PWM.

We will modify the board by removing the crystal. Hold the board with the crystal can facing down, and place the soldering iron so that both leads of the crystal simultaneously heat up enough to melt the solder. The crystal can will just fall out.

Now we can attach a wire where the crystal used to be. We want the hole closest to the center of the board. The other hole is already connected to ground, so we don’t need a wire for it.

While we have the soldering iron hot, we might as well attach the antenna. The holes on the side of the board away from the pins are the antenna (nearest the edge) and the ground (closer to the chip). I chose to solder in an SMA connector there, so I can easily switch antennas.

Photo by author

When we are all finished, and the antenna is connected and the receiver wired up to the ESP32, it looks like this (after I flattened things to improve the focus):

Press enter or click to view image in full size

Photo by author

For this project, I chose an ESP32 WROOM-32 (about $2 at AliExpress.com). The receiver gets its power and ground from the +5 volt and GND pins on the board. The data from the receiver goes to pin 36. The board has another ground pin near the middle, and the black wire from the speaker goes to that. The red wire of the speaker goes to pin 19. Lastly, the new pin on the receiver board that we added for tuning goes to pin 18.

The code looks like this:

From machine import Timer, Pin, PWM

Def main():

From time import sleep\_us

Freqs = [433.98, 315]

Speaker\_pin = Pin(19, Pin.OUT)

Speaker\_pin.value(0)

In\_pin = Pin(36, Pin.IN)

While True:

Speaker\_pin.value(0)

For x in freqs:

F = int(((x + (0.86 \* x/315))/64.5) \* 1000000)

Print(x, f)

Tune = PWM(Pin(18), freq=int(f), duty=512)

For count in range(1000000):

Speaker\_pin.value(in\_pin.value())

Sleep\_us(10)

Main()

We will be playing with two frequencies, 119 megahertz apart. These are the two frequencies you can buy the little FS1000a transmitters for, so that is convenient. The frequencies are 433.98 megahertz and 315 megahertz.

You can see we have attached the speaker to pin 19, and we will be copying the state of pin 36 to pin 19. We must do this because our little receiver board does not have enough power to drive a speaker (but the ESP32 pin does).

You may have noticed that the crystal can was marked 6.7458 and not 433.98. That is because the superheterodyne chip has a frequency multiplier built in. This allows for the use of crystals such as our 6,745,800 Hertz crystal, but it also allows little computers like the ESP32 to set the frequency, even though they can only reach 40 megahertz, and not the 433.98 required.

The complicated-looking arithmetic is just the way we convert the frequency we want (433.98 MHz) into the frequency the receiver wants. The receiver will then multiply the frequency we give it to the final 433.98 MHz.

We print out the frequency we asked for, and the frequency we send to the receiver.

Next, we set up the Pulse Width Modulator to send out square waves (duty cycle 512) at the right frequency on pin 18.

The loop after that spins a million times, copying the receiver output into the speaker and then waiting for 10 microseconds. The whole loop thus takes 10 seconds, during which time the receiver is listening to the first frequency, and we can hear some noise from the speaker.

In this way, our device will let us hear communications on first 433.98 MHz, and then on 315 MHz, and then back, over and over.

To make the demonstration interesting, we will build two of our little ESP8266 Morse by Keyboard projects. One will run the following code:

From morse import Morse

Def main():

PIN\_D4 = 2

# tone = int(input(”Tone? “))

Tone = 800

Morse = Morse(PIN\_D4, tone)

Print(”Morse code AM Beacon”)

Morse.speed(20)

# str = input(”Enter the message to send: “)

Str = “Patrizia knows two or three letters of morse code, but she won’t admit to it.”

While(True):

Morse.send(str)

Main()

The other ESP8266 beacon will be running this code:

From morse import Morse

Def main():

PIN\_D4 = 2

# tone = int(input(”Tone? “))

Tone = 300

Morse = Morse(PIN\_D4, tone)

Print(”Morse code AM Beacon”)

Morse.speed(20)

# str = input(”Enter the message to send: “)

Str = “Dave used to know some Morse code, but has forgotten all but a few letters.”

While(True):

Morse.send(str)

Main()

Since each beacon uses a different tone, it is easy to tell which one we are receiving. As our receiver switches between the two frequencies, we can hear the Morse code change tone and message. It sounds like there is a conversation going on.