Python Radio 9: A 40-meter CW transmitter

Simon Quellen Field

Simon Quellen Field

Follow

11 min read

·

Aug 26, 2024

Listen

Share

More

Using just the ESP32.

Press enter or click to view image in full size

Photo by author

Our little ESP32 can produce square waves at frequencies as high as 40 megahertz. But that comes with some limitations. Not all frequencies from zero to 40 MHz are actually possible.

Microprocessors use timer peripherals to produce the Pulse Width Modulation (PWM). These timers have a high-speed clock (in the case of our ESP32 that clock runs at 80 MHz) that they then “divide down”. Each time the clock ticks, a counter is decremented, and when it gets to zero, a pin changes state.

Since the clock can only be “divided” by integers, only certain frequencies are actually available to us.

In the amateur radio bands available to those with a license, those frequencies are:

160-meter band: 1802816, 1807909, 1813031, 1818181, 1823361, 1828571, 1833810, 1839080, 1844380, 1849710, 1855072, 1860465, 1865889, 1871345, 1876832, 1882352, 1887905, 1893491, 1899109, 1904761, 1910447, 1916167, 1921921, 1927710, 1933534, 1939393, 1945288, 1951219, 1957186, 1963190, 1969230, 1975308, 1981424, 1987577, 1993769, 2000000

80-meter band: 3506849, 3516483, 3526170, 3535911, 3545706, 3555555, 3565459, 3575418, 3585434, 3595505, 3605633, 3615819, 3626062, 3636363, 3646723, 3657142, 3667621, 3678160, 3688760, 3699421, 3710144, 3720930, 3731778, 3742690, 3753665, 3764705, 3775811, 3786982, 3798219, 3809523, 3820895, 3832335, 3843843, 3855421, 3867069, 3878787, 3890577, 3902439, 3914373, 3926380, 3938461, 3950617, 3962848, 3975155, 3987538, 4000000

40-meter band: 7013698, 7032967, 7052341, 7071823, 7091412, 7111111, 7130919, 7150837, 7170868, 7191011, 7211267, 7231638, 7252124, 7272727, 7293447, 7314285

30-meter band: 10118577, 10138613, 10158730

20-meter band: 14027397, 14065934, 14104683, 14143646, 14182825, 14222222, 14261838, 14301675, 14341736, 14382022

17-meter band: 18091872, 18156028, 18220640

15-meter band: 21026694, 21069958, 21113402, 21157024, 21200828, 21244813, 21288981, 21333333, 21377870, 21422594, 21467505

12-meter band: 24914841, 24975609, 25036674

10-meter band: 28054794, 28131868, 28209366, 28287292, 28365650, 28444444, 28523676, 28603351, 28683473, 28764044, 28845070, 28926553, 29008498, 29090909, 29173789, 29257142, 29340974, 29425287, 29510086, 29595375, 29681159, 29767441

Frequencies shown above in bold are available for CW to the entry-level classes: Novice, Technician, and Technician Plus.

I have omitted the 60-meter band since it is channelized and does not allow arbitrary frequencies.

The program for demonstrating these frequencies is here:

From machine import Pin, PWM

Def main():

Pin = Pin(18, Pin.OUT)

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

Bands = (

( 1800000, 2000000), # 160 meters

( 3500000, 4000000), # 80 meters

( 7000000, 7300000), # 40 meters

(10100000, 10150000), # 30 meters

(14000000, 14350000), # 20 meters

(18068888, 18168000), # 17 meters

(21000000, 21450000), # 15 meters

(24890000, 24990000), # 12 meters

(28000000, 29700000) # 10 meters

)

Guess = 0

For x in bands:

F\_lo, f\_hi = x

For f in range(f\_lo, f\_hi):

Pwm.freq(f)

Actual = pwm.freq()

If actual != guess:

Print(str(actual) + “, “, end=”)

Guess = actual

Print()

Main()

If you don’t (yet) have an amateur radio license, let me take this opportunity to encourage you to get one, since if you have read this far, you obviously have an interest, and the license is very easy to get (Google for “amateur radio license”). The tests are simple, all the questions and answers are available online, there are many free tutorial guides, and unlimited online practice tests, and you only have to get 70% of the questions right to pass.

That said, it is unlikely that you will get into trouble building our first transmitter, as it can only reach a few feet. If you have an amateur radio listener in the apartment next door, you might bother them, so we will place what we call a “dummy load” on our transmitter to ensure that it is not an “intentional transmitter” and will thus fall into the FCC’s Part 15 rules, and be allowed. A dummy load is just a resistor between the antenna pin and the ground pin. A nearby shortwave radio will allow you to hear the Morse code it sends, but it won’t leave the room.

Since we will be sending Morse code, we can re-use our Morse code program. But we will make a small change. Our original program modulated the transmitter with an audio signal so we could hear it on a speaker. But since we will be receiving our signal using a shortwave radio, we will instead be sending an unmodulated carrier wave. This mode is known as “continuous wave” or CW.

Our modified module looks like this:

From machine import Pin, PWM

Class CWMorse:

Character\_speed = 18

Def \_\_init\_\_(self, pin, freq):

Self.key = PWM(Pin(pin, Pin.OUT))

Self.key.freq(freq)

Def speed(self, overall\_speed):

If overall\_speed >= 18:

Self.character\_speed = overall\_speed

Units\_per\_minute = int(self.character\_speed \* 50) # The word PARIS is 50 units of time

OVERHEAD = 2

Self.DOT = int(60000 / units\_per\_minute) – OVERHEAD

Self.DASH = 3 \* self.DOT

Self.CYPHER\_SPACE = self.DOT

If overall\_speed >= 18:

Self.LETTER\_SPACE = int(3 \* self.DOT) – self.CYPHER\_SPACE

Self.WORD\_SPACE = int(7 \* self.DOT) – self.CYPHER\_SPACE

Else:

# Farnsworth timing from <https://www.arrl.org/files/file/Technology/x9004008.pdf>

Farnsworth\_spacing = (60000 \* self.character\_speed – 37200 \* overall\_speed) / (overall\_speed \* self.character\_speed)

Farnsworth\_spacing \*= 60000/68500 # A fudge factor to get the ESP8266 timing closer to correct

Self.LETTER\_SPACE = int((3 \* farnsworth\_spacing) / 19) – self.CYPHER\_SPACE

Self.WORD\_SPACE = int((7 \* farnsworth\_spacing) / 19) – self.CYPHER\_SPACE

Def send(self, str):

From the\_code import code

From time import sleep\_ms

For c in str:

If c == ‘ ‘:

Self.key.duty(0)

Sleep\_ms(self.WORD\_SPACE)

Else:

Cyphers = code[c.upper()]

For x in cyphers:

If x == ‘.’:

Self.key.duty(512)

Sleep\_ms(self.DOT)

Else:

Self.key.duty(512)

Sleep\_ms(self.DASH)

Self.key.duty(0)

Sleep\_ms(self.CYPHER\_SPACE)

Self.key.duty(0)

Sleep\_ms(self.LETTER\_SPACE)

The only change is in the \_\_init\_\_() method. Instead of using a fixed frequency of 300 Hertz, we pass in the frequency as a parameter to the method. This is because instead of modulating the carrier from an external transmitter, the ESP32 itself will be the actual transmitter.

Our main program looks like this:

From cwmorse import CWMorse

From machine import Pin, PWM

From time import sleep

Def main():

OUT\_PIN = 18

F = 7032966

Pin = Pin(18, Pin.OUT)

Pwm = PWM(pin, freq=f, duty=512) # So that we can read the actual frequency

Actual = pwm.freq()

Pwm.deinit()

Pwm = None

Cw = CWMorse(OUT\_PIN, f)

Cw.speed(20)

Print(”CW transmitter”)

Msg = “AB6NY testing ESP32 as a 40 meter transmitter sending on “ + str(actual) + “ Hertz.”

While True:

Print(msg)

Cw.send(msg)

Sleep(5)

Main()

We import our modified CWMorse module. We will use pin 18 for our PWM output pin, and we will transmit on a frequency of 7,032,967 Hertz. I chose to set the frequency to one Hertz below that, knowing that the chip would do the arithmetic and pick the next available frequency.

The next little bit of code is only there because we want to print out the actual frequency. We set up the PWM, get the actual frequency, and then tear it back down so that the pin is available to our CWMorse module.

The rest of the program should be familiar. We create the cw object, passing in the frequency. We set the Morse code speed to 20 words per minute. Then we loop, sending out the Morse-coded message, and then sleeping for 5 seconds before repeating. What we have built is what amateur radio operators call a beacon.

Make sure you change the call sign from mine (AB6NY) to your own before you transmit.

Our simple circuit looks like this:

Press enter or click to view image in full size

Image by author

The resistor value is not critical. I used 220 ohms. A 50-ohm dummy load is more common, as most transmitters are designed to drive 50-ohm loads. Our little computer was not designed to be a transmitter, so any value that will absorb the energy will do.

We have a problem before we can get on the air, however. Our transmitter is putting out square waves.

Press enter or click to view image in full size

Photo by author

Square waves are made up of a sine wave (called the fundamental) and every odd harmonic of the fundamental. So our 7,032,967 Hertz transmitter is also transmitting on 21,098,901 Hertz (at one-third the power), 35,164,838 Hertz (at one-fifth the power), 49,230,769 Hertz (at one-seventh the power), and so on.

This is not good. We don’t want to transmit on any frequency but one.

We can clean up our signal using a low-pass filter. It will only pass the low frequency, and block (or seriously reduce) all of the others.

A simple low-pass filter is just a coil and a capacitor. The coil is in series, and the capacitor is in parallel.

Press enter or click to view image in full size

Image by author

This is called an “L” network since the inductor and capacitor form an “L” shape.

Filters not only convert square waves to sine waves by removing harmonic frequencies, they can also be used to match the output impedance of a transmitter to the input impedance of an antenna. However, the simple “L” network can only match one impedance to another with one combination of inductance and capacitance. By adding a second capacitor, we can get more freedom in selecting parts values, and we get more control over how steeply the high frequencies are attenuated.

Press enter or click to view image in full size

Image by author

This configuration is known as a “Pi” filter since the inductor and two capacitors look like the Greek letter pi.

Finally, we don’t want any DC voltage getting to our antenna when the microprocessor is starting up. So we put a capacitor between the transmitter and the filter. This will look like there is no connection when the signal is not changing.

Press enter or click to view image in full size

The resulting oscilloscope trace of our waveform now looks much closer to a sine wave:

Press enter or click to view image in full size

Photo by author

The actual filter looks like this:

Press enter or click to view image in full size

Photo by author

The input and ground connections are at the bottom right. The antenna is the black insulated wire on the left. If you are using a coax cable to an external antenna, the shield would be soldered to the ground wire at the bottom left, and the center conductor would attach where the black insulated wire is.

To aid in calculating the inductance and capacitance, we can run the following program on the desktop computer:

Import math

Def humanify(value, label):

Index = 0

While value < 0.005:

Index += 1

Value \*= 1000000

Labels = [” “, “ u”, “ p”]

Value = round(value, 4)

Return str(value) + labels[index] + label

Def common\_capacitors(value):

Vals = [10, 22, 33, 47, 68, 100, 150, 220, 330, 470, 560,

1000, 2200, 3300, 4700, 6800, 10000, 22000, 33000, 47000, 100000, 1000000]

Old\_diff = 1000000000000

For x in vals:

Y = x / 1000000000000

Diff = abs(value – y)

If diff < old\_diff:

Return\_val = y

Old\_diff = diff

Return return\_val

Def common\_inductors(value):

Vals = [.1, .15, .47, .68, 1, 1.5, 2.2, 3.3, 4.7, 6.8, 8.2, 10, 15, 22, 33,

47, 68, 100, 120, 150, 220, 330, 470, 680, 1000, 10000, 22000, 33000, 47000, 68000, 100000]

Old\_diff = 1000000000000

For x in vals:

Y = x / 1000000

Diff = abs(value – y)

If diff < old\_diff:

Return\_val = y

Old\_diff = diff

Return return\_val

Def filter(input\_impedance, output\_impedance, Q, F):

If input\_impedance < output\_impedance:

Lo = input\_impedance

Hi = output\_impedance

Else:

Lo = output\_impedance

Hi = input\_impedance

newQ = Q

while newQ \* newQ + 1 <= hi / lo:

newQ += 0.1

squared = (hi / lo) / (newQ \* newQ + 1 – (hi / lo))

while squared <= 0:

newQ += 0.1

squared = (hi / lo) / (newQ \* newQ + 1 – (hi / lo))

print(”Squared:”, squared)

if newQ > Q:

Q = newQ

Print(”Boosted Q to”, Q, “to get the impedance to match”)

C2\_reactance = lo \* math.sqrt((hi / lo) / (Q \* Q + 1 – (hi / lo)))

C2 = 1 / (2 \* math.pi \* F \* C2\_reactance)

C1\_reactance = hi / Q

C1 = 1 / (2 \* math.pi \* F \* C1\_reactance)

L1\_reactance = (Q \* hi + (hi \* lo / C2\_reactance)) / (Q \* Q + 1)

L1 = L1\_reactance / (2 \* math.pi \* F)

Print()

Print(”C1:”, humanify(C1, “F”))

Print(”L1:”, humanify(L1, “H”))

Print(”C2:”, humanify(C2, “F”))

If abs(C1 – C2) < 0.0000001:

Print(”Output impedance:”, round(math.sqrt(L1 / C2)))

If C1 > C2:

Center\_frequency = 1 / (2 \* math.pi \* math.sqrt(L1 \* C1))

Cutoff\_frequency = 1 / (math.pi \* math.sqrt(L1 \* C1))

Else:

Center\_frequency = 1 / (2 \* math.pi \* math.sqrt(L1 \* C2))

Cutoff\_frequency = 1 / (math.pi \* math.sqrt(L1 \* C2))

Print(”Center frequency:”, round(center\_frequency))

Print(”3dB cutoff frequency:”, round(cutoff\_frequency))

C1 = common\_capacitors(C1)

C2 = common\_capacitors(C2)

L1 = common\_inductors(L1)

Print()

Print(”In closest common values:”)

Print(”C1:”, humanify(C1, “F”))

Print(”L1:”, humanify(L1, “H”))

Print(”C2:”, humanify(C2, “F”))

If abs(C1 – C2) < 0.0000001:

Print(”Output impedance:”, round(math.sqrt(L1 / C2)))

If C1 > C2:

Center\_frequency = 1 / (2 \* math.pi \* math.sqrt(L1 \* C1))

Cutoff\_frequency = 1 / (math.pi \* math.sqrt(L1 \* C1))

Else:

Center\_frequency = 1 / (2 \* math.pi \* math.sqrt(L1 \* C2))

Cutoff\_frequency = 1 / (math.pi \* math.sqrt(L1 \* C2))

Print(”Center frequency:”, round(center\_frequency))

Print(”3dB cutoff frequency:”, round(cutoff\_frequency))

Def main():

Print()

Print(”For example, try 50, 50, 1, 7000000”)

Print()

Input\_impedance = float(input(”Input impedance? “))

Output\_impedance = float(input(”Output impedance? “))

Q = float(input(”Q? “))

F = float(input(”Center frequency? “))

Filter(input\_impedance, output\_impedance, Q, F)

Main()

For our filter, we input 50 ohms for the input and output impedances, 1 for the Q, and 7032967 for the center frequency. We get:

For example, try 50, 50, 1, 7000000

Input impedance? 50

Output impedance? 50

Q? 1

Center frequency? 7032967

C1: 452.5969 pF

L1: 1.1315 uH

C2: 452.5969 pF

Output impedance: 50

Center frequency: 7032967

3dB cutoff frequency: 14065934

In closest common values:

C1: 470.0 pF

L1: 1.0 uH

C2: 470.0 pF

Output impedance: 46

Center frequency: 7341270

3dB cutoff frequency: 14682540

Note that although the frequency came out a little higher than we asked for when we used standard values, and the output impedance is not quite 50 ohms, the results are close enough to make a decent filter. You can use online programs to design filters with more elements if you want even better performance.

A low-pass filter takes advantage of the properties of coils and capacitors. If we put direct current across a capacitor, it will charge up to its capacitance, but then it looks like an open switch, and no energy passes through. But at high frequencies, the capacitor charges and discharges rapidly, and the alternating current passes through as if the switch were closed.

A coil has the opposite characteristic. It lets direct current through once it has created the magnetic field around it. But that magnetic field collapses when the current is turned off or reversed, and as it collapses, it generates current in the coil, keeping the current flowing in the same direction it had been. So a coil resists changes in the current.

In our filter, low frequencies are barely impeded by our tiny coil. It does not build up much of a magnetic field because it is just a few turns of wire. Since the transmitter and the antenna are directly connected to the coil, low frequencies pass right on through.

High frequencies are blocked by the coil. But we also have the two capacitors. They act like short circuits to ground for high frequencies. High frequencies do not make it out of the antenna.