# Python Radio 12: An AM Transmitter

[Simon Quellen Field](https://medium.com/@simon.field_37276?source=post_page---byline--6326cc547ae8---------------------------------------)

[Simon Quellen Field](https://medium.com/@simon.field_37276?source=post_page---byline--6326cc547ae8---------------------------------------)

Follow

4 min read

·

Aug 30, 2024

Listen

Share

More

Using the RP2040 microcontroller

Press enter or click to view image in full size



MidJourney

We can easily turn the RP2040 into an AM radio transmitter. It is capable of transmitting AM signals anywhere in the entire AM radio band, and also (with a license) in any of the HF amateur radio bands.

The first step is to generate a square wave on the frequency to which we will tune the radio. In the example here, we will use 540 kilohertz. We will output that on pin 15 of the RP2040.

If we connect pin 15 to either a long wire antenna or (more easily) to a good ground, such as the metal case of some equipment that is plugged into the wall, or the screw that holds the outlet cover or switch cover to the wall, then the signal will be heard clearly throughout the building without any further amplification.

With just the carrier, all we will hear on the AM radio is that the static suddenly gets quiet when we start transmitting.

To AM modulate the signal, we can simply add a PWM tone on another pin (such as pin 14) and connect that to pin 15. Now we hear that tone on the radio.

Of course, we can then send Morse code using that tone. Here is the code to do that, starting with the main routine:

from cwmorse import CWMorse  
from time import sleep  
  
frequency = 540000  
tone = 14  
carrier = 15  
  
def main():  
 cw = CWMorse(carrier, tone, frequency)  
 cw.speed(10)  
 print(”CW transmitter”)  
 msg = “AB6NY testing RP2040 as an AM transmitter sending on “ + str(frequency) + “ Hertz.”  
 cw.tune(True)  
 sleep(30)  
 cw.tune(False)  
   
 while True:  
 print(msg)  
 cw.send(msg)  
 sleep(5)  
  
main()

The cwmorse.py module looks familiar, with just a few changes:

from machine import Pin, PWM  
from time import sleep  
  
class RP\_CW:  
 def \_\_init\_\_(self, carrier\_pin, tone\_pin, freq):  
 from machine import Pin  
 from rp2 import PIO, StateMachine, asm\_pio  
 self.tone\_pin = Pin(tone\_pin, Pin.OUT)  
 self.carrier\_pin = Pin(carrier\_pin, Pin.OUT)  
 self.pwm = PWM(tone\_pin, freq=500, duty\_u16=0)  
   
 @asm\_pio(set\_init=PIO.OUT\_LOW)  
 def square():  
 wrap\_target()  
 set(pins, 1)  
 set(pins, 0)  
 wrap()  
  
 self.f = freq  
 self.sm = StateMachine(0, square, freq=2\*self.f, set\_base=self.carrier\_pin)  
 self.sm.active(1)  
  
 def on(self):  
 self.pwm.duty\_u16(32767)  
  
 def off(self):  
 self.pwm.duty\_u16(0)  
   
 def frequency(self, frq):  
 self.f = frq  
  
class CWMorse:  
 character\_speed = 20  
  
 def \_\_init\_\_(self, carrier\_pin, tone\_pin, freq):  
 self.cw = RP\_CW(carrier\_pin, tone\_pin, freq)  
 self.cw.frequency(freq)  
   
 def tune( self, onoff):  
 if onoff:  
 self.cw.on()  
 else:  
 self.cw.off()  
   
 def speed(self, overall\_speed):  
 if overall\_speed >= 20:  
 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 >= 20:  
 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.cw.off()  
 sleep\_ms(self.WORD\_SPACE)  
 else:  
 cyphers = code[c.upper()]  
 for x in cyphers:  
 if x == ‘.’:  
 self.cw.on()  
 sleep\_ms(self.DOT)  
 else:  
 self.cw.on()  
 sleep\_ms(self.DASH)  
 self.cw.off()  
 sleep\_ms(self.CYPHER\_SPACE)  
 self.cw.off()  
 sleep\_ms(self.LETTER\_SPACE)

We have added a tone pin and changed the on() and off() methods to turn on and off the PWM on that pin. The rest of the module is unchanged.

Using the household wiring ground allows us to hear the signal throughout the building, without much of the signal escaping to bother anyone next door. The low power levels and lack of an antenna make the device legal to operate.

If you like, you can modify the code to use the ringtones from our previous project to send annoying music over the radio.

We can, however, arrange to send less annoying signals. I have an antique AM radio, and I like to have it play old-time radio shows instead of modern AM radio broadcasts.

On the Internet, you can find many MP3 recordings of old-time radio shows. Your computer has an audio jack where speakers or earphones can be connected. We connect the shield of the phone plug to the ground of the RP2040, and the tip of the phone plug to pin 15. We also connect the anode of a 1N4148 diode to pin 15. The cathode (the side of the diode that has the black band) connects to a good ground.

The diode acts as a modulator, allowing the sound from the computer to vary the amplitude of the signal from the RP2040.

My various modern AM radios can hear the signal clearly anywhere in the house. My antique radio picks up too much noise if it is farther away than about four feet from the transmitter. This may be because its cord does not have a ground wire. Still, the transmitter can hide in a drawer, and the radio can sit on the cabinet above the drawer and play old radio shows. The Lone Ranger, Fibber McGee and Molly, the CBS Mystery Show, Jack Benny, and even radio news recordings of the Hindenburg disaster or the Pearl Harbor attack.

The schematic looks like this:

Press enter or click to view image in full size

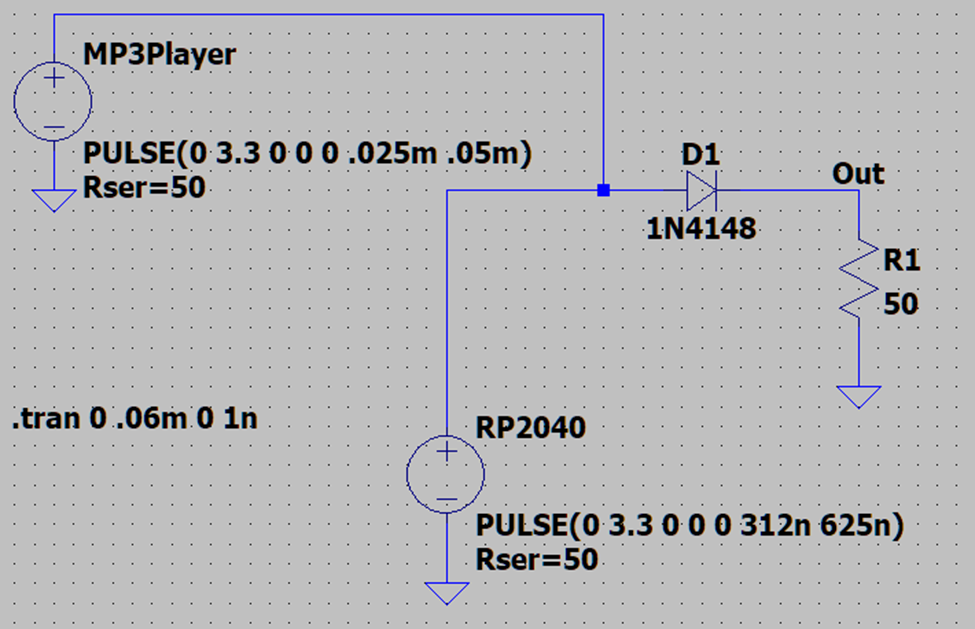


Image by author

If the MP3 player was emitting a 16 kilohertz square wave, and the RP2040 was putting out a 1600 kilohertz carrier, the modulated waveform would look something like this:

Press enter or click to view image in full size

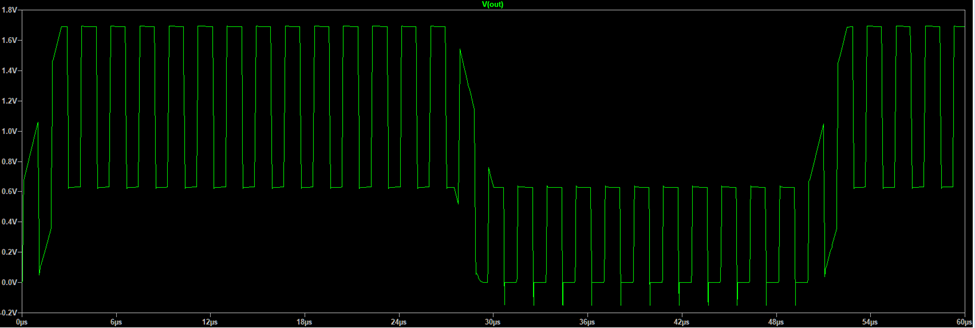


Image by author