Python Radio 43: Super High Frequency Radar

Simon Quellen Field

Simon Quellen Field

Follow

5 min read

·

Jun 16, 2025

Listen

Share

More

24 Gigahertz radar in the 1.25 centimeter band

Press enter or click to view image in full size

24 GHz Radar

All photos by the author.

In the previous chapter on radar, we used a device that operated at 3 GHz, resulting in a 10-centimeter wavelength. But we can now buy radar devices for less than $10 that deliver 24 gigahertz. That’s a wavelength of 1.25 centimeters, placing it in the middle of the K band (Super High Frequency, or SHF).

The simplest version, shown in the photo above, is very easy to use. It sends the range in centimeters via serial at 115200 baud.

I chose the ESP32-C3 Super Mini with the tiny OLED display for this part of the project. Here is the micropython code:

From machine import UART, I2C, Pin

From SH1106 import SH1106\_I2C

From sys import print\_exception

From time import sleep\_ms

Class Display:

Def \_\_init\_\_(self):

Self.i2c\_display = I2C(0, sda=Pin(5), scl=Pin(6), freq=400\_000)

Self.display = SH1106\_I2C(128, 64, self.i2c\_display, rotate=180)

Self.display.contrast(255)

BufferWidth, BufferHeight = 128, 64

ScreenWidth, ScreenHeight = 72, 40

Self.xOffset, self.yOffset = (BufferWidth – ScreenWidth) // 2, (BufferHeight – ScreenHeight) // 2

# self.display.rotate(1)

Def oled(self, s, line, column):

Self.display.fill\_rect(self.xOffset, self.yOffset + 2 + line \* 9, 128-self.xOffset, 9, 0)

Self.display.text(s, self.xOffset + 2 + column \* 8, self.yOffset + 2 + line \* 9, 1)

Self.display.show()

D = None

Try:

D = Display()

Except OSError as e:

If e == 19: # ENODEV

D = None

# d.display.invert(0)

Uart = UART(1, baudrate=115200, tx=1, rx=2)

Def main():

If d:

d.oled(“Range:”, 0, 0)

d.oled(“cm”, 2, 0)

cm = “”

while True:

sleep\_ms(1)

if uart.any():

line = uart.readline()

if line:

try:

cm = line.decode(“utf-8”)

except Exception as e:

print\_exception(e)

print(“Line:”, line)

if “Range” in cm:

cm = cm[6:].strip()

print(cm)

if d:

d.oled(cm, 1, 0)

main()

The first half of the code handles the display, which we first introduced in Chapter 35 when we built the mailbox alarm.

The rest of the code loops, getting a line from the UART and parsing it to get the range to the target (a person in the typical use case for an automatic door opener).

When I took the picture, I was 9 centimeters from the device.

The simple device has one transmit antenna and one receive antenna. But the HLK-LD2450 device also comes in a package with two receiving antennas. That allows it to tell us the distance (the Y direction) and how far to the left or right the target is (the X direction).

Press enter or click to view image in full size

Radar with two receiving antennas.

Instead of using micropython, we will use regular Python on a Windows computer for a change. Linux or Macintosh will also work, using a different UART port name.

In the photo, you can see a USB-to-Serial module attached to the HLK-LD2450. We are only using one serial line, as we won’t be sending commands to the radar, only receiving data. The TX pin connects to the RX pin on the serial adapter. 3.3-volt power and ground complete the connections.

This device has a more complex data stream than the first one. It sends the bytes raw (not ASCII), in frames that have a header and end codes, and require a little decoding.

From time import sleep

From serial import Serial

Import matplotlib.pyplot as plt

Uart = Serial(“com7”, 256000)

Def convert\_sign(x):

If x & 0x8000:

X = x – 0x8000

Else:

X = -x

Return x

Def main():

Fig = plt.figure(figsize=(7, 7))

Plt.ion()

Plt.show()

While True:

Sleep(0.001)

Line = uart.read(30)

If line and len(line) >= 29:

# print(f”{hex(int.from\_bytes(line, ‘little’))}”)

Header = int.from\_bytes(line[:4], ‘little’)

A\_x = convert\_sign(int.from\_bytes(line[4:6], ‘little’))

A\_y = convert\_sign(int.from\_bytes(line[6:8], ‘little’))

A\_speed = convert\_sign(int.from\_bytes(line[8:10], ‘little’))

A\_resolution = int.from\_bytes(line[10:12], ‘little’)

B\_x = convert\_sign(int.from\_bytes(line[12:14], ‘little’))

B\_y = convert\_sign(int.from\_bytes(line[14:16], ‘little’))

B\_speed = convert\_sign(int.from\_bytes(line[16:18], ‘little’))

B\_resolution = int.from\_bytes(line[18:20], ‘little’)

C\_x = convert\_sign(int.from\_bytes(line[20:22], ‘little’))

C\_y = convert\_sign(int.from\_bytes(line[22:24], ‘little’))

C\_speed = convert\_sign(int.from\_bytes(line[24:26], ‘little’))

C\_resolution = int.from\_bytes(line[26:28], ‘little’)

End\_frame = int.from\_bytes(line[28:], ‘little’)

Plt.clf()

Fig.text((500+a\_x)/1000, a\_y/5000, f”o”)

Fig.text(0, 0, f”{a\_x} mm, {a\_y} mm, {a\_speed} cm/sec”)

Fig.canvas.draw()

Plt.pause(.02)

Plt.show()

# print(f”{hex(header)}”)

Print(f”X: {a\_x} mm”, end=” “)

Print(f”Y: {a\_y} mm”, end=” “)

Print(f”Speed: {a\_speed} cm/sec”, end=” “)

Print(“\r”, end=””)

# print(f”Resolution: {hex(a\_resolution)}, {a\_resolution}”)

# print(f”X: {hex(b\_x)}, {b\_x} mm”)

# print(f”Y: {hex(b\_y)}, {b\_y} mm”)

# print(f”Speed: {hex(b\_speed)}, {b\_speed} cm/sec”)

# print(f”Resolution: {hex(b\_resolution)}, {b\_resolution} mm”)

# print(f”X: {hex(c\_x)}, {c\_x} mm”)

# print(f”Y: {hex(c\_y)}, {c\_y} mm”)

# print(f”Speed: {hex(c\_speed)}, {c\_speed} cm/sec”)

# print(f”Resolution: {hex(c\_resolution)}, {c\_resolution} mm”)

# print(f”{hex(end\_frame)}”)

Main()

The manual gives an example to make decoding easier. Here is a sample frame:

AA FF 03 00 0E 03 B1 86 10 00 40 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 55 CC

The first four bytes AA FF 03 00 are the header. Every frame starts with this.

The last two bytes 55 CC are the end-of-frame marker. Every frame ends with this.

In between are three groups of eight bytes (0E 03 B1 86 10 00 40 01, 00 00 00 00 00 00 00 00, and 00 00 00 00 00 00 00 00). Each one is data for a target. The device can track three targets at once. In the example, there is only one target. The other two groups are all zeroes.

The data for a target has four 16-bit numbers, in “little-endian” format, that describe the X coordinate in millimeters, the Y coordinate in millimeters, the speed in centimeters per second, and the resolution (always 320 millimeters).

The sign bit is backwards from what we see in normal two’s-complement arithmetic, with a 0 meaning negative and a 1 meaning positive. We handle this unusual format with a convert\_sign() subroutine. The resolution (weirdly) does not use the sign bit convention.

Because we are running on a computer with a screen, we can plot the result using Matplotlib in Python. We set interactive mode (plt.ion()), adjust the coordinates to screen coordinates, and draw a little “o” to represent the person being tracked.

Since I am working alone here in the lab, I commented out the code for the other two targets.

Here is what it looks like:

The 24 GHz Radar screen in action.

The distance resolution claims about one foot (32 centimeters), but the data from the device shows single-centimeter resolution, which I take with about 320 grains of salt. But averaging a few hundred readings might get us a pretty accurate distance, perhaps with centimeter resolution.

The range of the device is about 8 meters (about 25 feet). It is most sensitive to human bodies, perhaps by ignoring things that don’t move. It might even be possible to use it to detect a heartbeat with a little digital signal processing.