Python Radio 46: Satellites and Atomic Time

Simon Quellen Field

Simon Quellen Field

Follow

5 min read

·

Jun 30, 2025

Listen

Share

More

A GPS Disciplined NTP Server in Micropython

Press enter or click to view image in full size

A GPS satellite over North America.

MidJourney

For some weak signal protocols, such as WSPR, we need a clock that keeps good time. Anyone wishing to receive our signal knows to start receiving at an exact time, and if our clock doesn’t match theirs, we can’t communicate.

When we have an Internet connection, this is easy. We just use an NTP server to get accurate time from the web, and set our computer’s clock by that.

But if we are camping out in the wilderness, we will need something else.

This project uses an inexpensive LilyGo T-Beam microcomputer running Micropython to collect GPS data from the satellites and become an NTP server over its Wi-Fi radio for any other computers within range.

The T-Beam has a GPS receiver built in. It also has Wi-Fi. And becoming an NTP server is surprisingly easy.

Here is the code:

From machine import Pin, I2C, RTC, UART, freq

From time import sleep, sleep\_ms, time

From network import WLAN, AP\_IF

From socket import socket, AF\_INET, SOCK\_DGRAM

From struct import pack\_into

From micropyGPS import MicropyGPS

Freq(240\_000\_000)

Last\_rtc\_sync\_time = 0

RTC\_SYNC\_INTERVAL\_SECONDS = 6 \* 3600

NTP\_PORT = 123

NTP\_DELTA = 3155673600

Def setup\_wifi\_ap():

WIFI\_SSID, WIFI\_PASS = “T-Beam Time Server”, “gps-time-rocks”

Ap = WLAN(AP\_IF)

Ap.active(True)

Ap.config(essid=WIFI\_SSID, password=WIFI\_PASS)

While not ap.active():

Sleep(1)

Print(f”--- Wi-Fi Access Point Started: SSID={WIFI\_SSID}, IP={ap.ifconfig()[0]} ---“)

Def calculate\_weekday(y, m, d):

If m < 3:

M += 12

Y -= 1

T = [0, 3, 2, 5, 0, 3, 5, 1, 4, 6, 2, 4]

Day = (y + y // 4 – y // 100 + y // 400 + t[m – 1] + d) % 7

Return day if day != 0 else 7

Def set\_rtc\_from\_gps(gps\_object):

Global last\_rtc\_sync\_time

Is\_initial\_sync = (last\_rtc\_sync\_time == 0)

Time\_since\_last\_sync = time() – last\_rtc\_sync\_time

Should\_sync = is\_initial\_sync or (time\_since\_last\_sync > RTC\_SYNC\_INTERVAL\_SECONDS)

If should\_sync and gps\_object.date[0] != 0:

Y, m, d = gps\_object.date[2] + 2000, gps\_object.date[1], gps\_object.date[0]

H, mn, s = gps\_object.timestamp[0], gps\_object.timestamp[1], int(gps\_object.timestamp[2])

Dt = (y, m, d, calculate\_weekday(y, m, d), h, mn, s, int((gps\_object.timestamp[2] – s) \* 1e6))

RTC().datetime(dt)

Last\_rtc\_sync\_time = time()

Msg = “Initial RTC sync” if is\_initial\_sync else “RTC re-synchronized”

Print(f”\nINFO: {msg}. Current Time: {y}-{m:02d}-{d:02d} {h:02d}:{mn:02d}:{s:02d} UTC\n”)

Def main():

GPS\_RX=34

GPS\_TX=12

Setup\_wifi\_ap()

Uart = UART(2, baudrate=9600, tx=GPS\_TX, rx=GPS\_RX, timeout=10)

My\_gps = MicropyGPS(location\_formatting=’dd’)

Ntp\_socket = socket(AF\_INET, SOCK\_DGRAM)

Ntp\_socket.bind((‘’, NTP\_PORT))

Ntp\_socket.setblocking(False)

Cnt = 0

Lat\_avg = 0

Lon\_avg = 0

Alt\_avg = 0

While True:

If uart.any():

If uart\_data := uart.read():

For char\_byte in uart\_data:

My\_gps.update(chr(char\_byte))

If my\_gps.satellites\_in\_use > 0 and my\_gps.latitude[0] != 0.0:

Set\_rtc\_from\_gps(my\_gps)

Lat\_data, lon\_data = my\_gps.latitude, my\_gps.longitude

Lat\_val, lon\_val = lat\_data[0], lon\_data[0]

If lat\_data[1] == ‘S’:

Lat\_val = -lat\_val

If lon\_data[1] == ‘W’:

Lon\_val = -lon\_val

Alt\_val, sats = my\_gps.altitude, my\_gps.satellites\_in\_use

If lat\_avg == 0: lat\_avg = lat\_val

If lon\_avg == 0: lon\_avg = lon\_val

If alt\_avg == 0: alt\_avg = alt\_val

If cnt < 20:

Cnt += 1

Lat\_avg = (lat\_avg \* (cnt-1) + lat\_val) / cnt

Lon\_avg = (lon\_avg \* (cnt-1) + lon\_val) / cnt

Alt\_avg = (alt\_avg \* (cnt-1) + alt\_val) / cnt

Feet = alt\_avg \* 3.280839895

Timestamp = f”{my\_gps.timestamp[0]:02d}:{my\_gps.timestamp[1]:02d}:{my\_gps.timestamp[2]:02.0f}”

Payload\_str = f”{lat\_avg:.9f}, {lon\_avg:.9f}, {alt\_avg:.5f} meters ({feet:.5f} feet), UTC: {timestamp}, {sats:3d} satellites”

Print(f”{payload\_str}”)

Sleep\_ms(300)

Else:

Print(f”Waiting for GPS fix... Sats: {my\_gps.satellites\_in\_view}”)

Sleep(3)

If last\_rtc\_sync\_time != 0:

Try:

Data, addr = ntp\_socket.recvfrom(48)

If data:

Print(f”NTP Request from {addr[0]}”)

Recv\_timestamp = time() + NTP\_DELTA

Ntp\_response = bytearray(48)

Ntp\_response[0] = 0x24

Ntp\_response[24:32] = data[40:48]

Secs, frac = int(recv\_timestamp), int((recv\_timestamp % 1) \* (2\*\*32))

Pack\_into(‘!II’, ntp\_response, 32, secs, frac)

Pack\_into(‘!II’, ntp\_response, 40, secs, frac)

Ntp\_socket.sendto(ntp\_response, addr)

Except OSError as e:

If e.args[0] != 11:

Print(f”NTP Socket Error: {e}”)

Main()

Starting in main(), we tell the program where to find the GPS receive and transmit pins. The GPS chip on the T-Beam connects to the UART at 9600 baud.

The MicropyGPS class knows how to parse the cryptic GPS sentences that come over the UART and turn them into things we can use. More on that later.

Then we bind a socket to the NTP port 123 and set it to non-blocking. This is how we will talk to NTP clients.

The while loop looks for data on the UART, and hands each byte to the update() method of MicropyGPS to parse and keep track of.

If the GPS has seen any satellites and has latitude data for us, we set the T-Beam real-time clock (RTC) from that data.

GPS latitude, longitude, and especially altitude are not perfectly accurate. We average the last 20 readings to get more precision.

Then we print out the data.

If we have synced our clock with the GPS satellite’s atomic clock, we are ready to serve NTP clients.

We get the request (if there is one) from the socket, and prepare our reply.

The setup\_wifi\_ap() routine is very simple, mostly boilerplate. You can set the SSID and password to suit your taste.

The set\_rtc\_from\_gps() routine makes sure we don’t set our clock too often (which can cause jitter problems for some applications). In Micropython, the weekday is ignored, but we include it here for completeness.

The micropyGPS.py module comes from GitHub: <https://github.com/inmcm/micropyGPS>. Download it and copy it to the T-Beam along with main.py.

When you boot the T-Beam, it starts printing to the Putty window (the terminal emulator I use on Windows):

37.203254700, -122.008956909, 487.72998 meters (1600.16394 feet), UTC: 22:54:26, 10 satellites

37.203254700, -122.008956909, 487.67343 meters (1599.97852 feet), UTC: 22:54:27, 10 satellites

37.203254700, -122.008956909, 487.61972 meters (1599.80225 feet), UTC: 22:54:27, 10 satellites

37.203254700, -122.008956909, 487.56870 meters (1599.63489 feet), UTC: 22:54:27, 10 satellites

37.203254700, -122.008956909, 487.51031 meters (1599.44336 feet), UTC: 22:54:28, 10 satellites

37.203254700, -122.008956909, 487.45483 meters (1599.26123 feet), UTC: 22:54:28, 10 satellites

NTP Request from 192.168.4.2

37.203254700, -122.008956909, 487.40210 meters (1599.08826 feet), UTC: 22:54:28, 10 satellites

37.203254700, -122.008956909, 487.34198 meters (1598.89099 feet), UTC: 22:54:29, 10 satellites

37.203254700, -122.008956909, 487.28491 meters (1598.70374 feet), UTC: 22:54:29, 10 satellites

37.203254700, -122.008956909, 487.23065 meters (1598.52576 feet), UTC: 22:54:29, 10 satellites

37.203254700, -122.008956909, 487.17914 meters (1598.35681 feet), UTC: 22:54:30, 10 satellites

NTP Request from 192.168.4.2

37.203254700, -122.008956909, 487.13019 meters (1598.19617 feet), UTC: 22:54:30, 10 satellites

37.203254700, -122.008956909, 487.08368 meters (1598.04358 feet), UTC: 22:54:30, 10 satellites

37.203254700, -122.008956909, 487.03949 meters (1597.89856 feet), UTC: 22:54:31, 10 satellites

37.203254700, -122.008956909, 486.99750 meters (1597.76086 feet), UTC: 22:54:31, 10 satellites

37.203254700, -122.008956909, 486.95761 meters (1597.63000 feet), UTC: 22:54:31, 10 satellites

37.203254700, -122.008956909, 486.90973 meters (1597.47290 feet), UTC: 22:54:32, 10 satellites

Notice how the latitude and longitude quickly converge, but the altitude is still fluctuating. In addition to the location, the code prints out the Coordinated Universal Time (UTC). The acronym looks dyslexic to English speakers, but it matches most languages’ abbreviations for Universal Time.

UTC does not have daylight saving time, and reflects the time at the zero degree meridian, as it replaces the older Greenwich Mean Time (GMT).

In the printout above, you can see two NTP requests from my laptop computer. I logged into the T-Beam’s Wi-Fi and got the address 192.168.4.2. The T-Beam’s address is 192.168.4.1 (the default IP address for the access point server).

In Windows, you use the command net start w32time to start the NTP client on your machine. Then you type:

W32tm /config /manualpeerlist:”192.168.4.1” /syncfromflags:manual /reliable:yes /update

The T-Beam sees the request and sends the reply, and the NTP client on the laptop updates its clock.

Now my laptop computer can be out in the woods and still get the correct time when the UTC system decides we need another leap second.