**UGA Motorsports Data Acquisition System**

Contents

**No table of contents entries found.**

**Accuracy without PPS at room temperature**

Every minute 0.36ms aka every hour 21.6ms

1ms accuracy with lap times <1minute or with gps

10ms accuracy with lap times >1minute and no gps

# Arduino Wiring

A diagram of a computer chip

AI-generated content may be incorrect.

A computer chip with many colored wires

AI-generated content may be incorrect.

# Planning out the whole system

The Timing Gate will have a Wifi Antenna, GPS Module, and Beam Break trigger. The Timing gate sends and receives TCP messages on port 5000. Sends ETC time of the gate event. Can be paired with multiple gates that also send their trigger time. Each is identified by “Serial” number. Server side software logs the raw events in a file for autosave ability and calculates sector and lap times from the raw events. Simple labels in a menu will show connected devices and allow to select where it is placed. Maybe in the future it’ll show gate locations.

In the same application I would also like to view the live telemetry of the car. This will consist of a UDP stream of filtered can data. This is in hopes to have a little lag as possible, and save compute for other functions. This module will also have a GPS antenna that will sync the internal clock and send the GPS coordinates of the car to the server using the same UDP connection. The Server will log both of these data streams and save to a file. Ideally the server would also graph the coordinates relative to the gates. The server will also send the Timing Gate times and any flags to the car to be transmitted over CAN to the vehicle’s dashboard. Flags will be transmitted as a simple integer.

0 – Green

1 – Yellow

2 – Red

3 – Checkered

4 – Black

Lap time will be sent in seconds and milliseconds. Both will only be sent on an update. Eventually this will also be able to send commands to change settings on certain devices in the car.

Eventually the wireless link might get switched to a lora system or 900mhz and just use wifi to create a high speed uplink. The live telemetry module could also work with just the long range link and then I have a separate data logger to micro sd card module that has wifi for high speed log starting, stopping, and file transfer.

# Links 2 stuff

Esp32 with removeable antenna - <https://www.digikey.com/en/products/detail/olimex-ltd/ESP32-S2-WROVER-DEVKIT-LIPO-EA/21662599?gad_campaignid=20243136172>

Potential USB c to battery charger circuitry

ILTM Throughput calculations

Can Bus 500kbps

Spi Bus 8,000kbps

Can message 8 bytes + overhead, call is 128bit per message

500k/128 = 3.9k messages a second (Currently sending ~500 messages per second)

| **Layer** | **Max throughput (8-byte CAN payload)** |
| --- | --- |
| CAN bus 500 kbps | ~3,900 msg/sec |
| SPI 8 MHz + SD | ~62,500 msg/sec |
| WiFi UDP 2.4GHz | ~68,000 msg/sec |

# Improving Data Acquisition for the UGA Motorsports IC Team

The UGA Motorsports formula team needed a way to accurately time and car during events such as Acceleration and Autox as well as be able to reliably view vehicle data while driving. Currently the team is using a stopwatch for timing which is quite inaccurate. Not only is the human reaction time slow and inconsistent, but the person cannot be at both the start and finish line, so the point at which they think the car crosses the line could, and normally is, inaccurate. A simple solution to this problem is a beam break timing system, but commercial products are often prohibitively expensive or don’t have the capability to do what the team needs. It would be beneficial to the team if the timing gates were able to be easily combined with engine data for easier analysis after the fact. Additionally, the current live telemetry system on the vehicle is dependent on the ECU, which limits the amount of data and speed at which it can be transmitted. The current system is also not very tolerant of bad connections, leading to it frequently going out.

I propose a system that combines all data acquisition into one convenient system/application that allows data to be quickly saved, viewed, synchronized, and expanded. A vehicle telemetry link capable of up to 2km (The max distance from the pit at Michigan International Speedway) for key engine statistics and timing gates capable of a typical acceleration test, or multiple sectors in an Autox test. Additionally, it should be a convenient way to take driver notes, record tire data, and organize test data. This will make testing documentation easier and improve the quality of data collected. Further, it will allow for easier analysis and review of testing data.

The core principle linking each device together is accurate GPS timing. All data will be sent with the UTC time accurate to the millisecond. This will used to calculate the time between gate triggers and to combine the engine data with those timing events. When GPS is lost, network time protocol will be used to synchronize the devices.

Below is the initial GUI mockup for the application.

# Visual Mockup

A screenshot of a computer

AI-generated content may be incorrect.

Component/Functionality DiagramA whiteboard with black text and words

AI-generated content may be incorrect.

# NTP Backup for GPS PPS

Raw NTP Sync data from Arduino serial

Recorded on my local home network using an Att router. Ping times were verified

23:30:15.563 -> Sync: offset = 1761103532278201800 ns (1761103532278.202 ms), delay = 7.950 ms

23:30:15.833 -> 03:30:15.807036

23:30:16.592 -> T1: 284276011000, T2: 1761103816559430000, T3: 1761103816559453900, T4: 284286064000

23:30:16.592 -> Sync: offset = 1761103532278206970 ns (1761103532278.207 ms), delay = 10.029 ms

23:30:16.819 -> 03:30:16.808414

23:30:17.600 -> T1: 285288014000, T2: 1761103817570754000, T3: 1761103817570793000, T4: 285297027000

23:30:17.600 -> Sync: offset = 1761103532278255300 ns (1761103532278.255 ms), delay = 8.974 ms

23:30:17.817 -> 03:30:17.809263

23:30:18.615 -> T1: 286299014000, T2: 1761103818581988900, T3: 1761103818582027700, T4: 286308028000

23:30:18.615 -> Sync: offset = 1761103532278233920 ns (1761103532278.234 ms), delay = 8.975 ms

23:30:18.815 -> 03:30:18.810491

23:30:19.604 -> T1: 287310011000, T2: 1761103819596789000, T3: 1761103819596823900, T4: 287323074000

23:30:19.604 -> Sync: offset = 1761103532278687040 ns (1761103532278.687 ms), delay = 13.028 ms

23:30:19.832 -> 03:30:19.813267

23:30:20.647 -> T1: 288325011000, T2: 1761103820608378700, T3: 1761103820608420200, T4: 288335773000

23:30:20.647 -> Sync: offset = 1761103532278683230 ns (1761103532278.683 ms), delay = 10.720 ms

23:30:20.816 -> 03:30:20.812011

23:30:21.654 -> T1: 289337014000, T2: 1761103821618787800, T3: 1761103821618828300, T4: 289345039000

23:30:21.654 -> Sync: offset = 1761103532278558650 ns (1761103532278.559 ms), delay = 7.984 ms

23:30:21.847 -> 03:30:21.812791

23:30:22.666 -> T1: 290347014000, T2: 1761103822628444800, T3: 1761103822628476400, T4: 290356047000

23:30:22.666 -> Sync: offset = 1761103532278294070 ns (1761103532278.294 ms), delay = 9.001 ms

23:30:22.817 -> 03:30:22.812934

23:30:23.668 -> T1: 291358011000, T2: 1761103823640724400, T3: 1761103823640763400, T4: 291367107000

23:30:23.668 -> Sync: offset = 1761103532278233590 ns (1761103532278.234 ms), delay = 9.057 ms

23:30:23.833 -> 03:30:23.815194

23:30:24.680 -> T1: 292369014000, T2: 1761103824653490900, T3: 1761103824653530500, T4: 292380082000

23:30:24.680 -> Sync: offset = 1761103532277973340 ns (1761103532277.973 ms), delay = 11.028 ms

23:30:24.815 -> 03:30:24.816972

23:30:25.682 -> T1: 293382014000, T2: 1761103825669039100, T3: 1761103825669058000, T4: 293396091000

23:30:25.682 -> Sync: offset = 1761103532278371060 ns (1761103532278.371 ms), delay = 14.058 ms

23:30:25.847 -> 03:30:25.819006

23:30:26.687 -> T1: 294398014000, T2: 1761103826679729600, T3: 1761103826679755700, T4: 294409548000

23:30:26.687 -> Sync: offset = 1761103532278007080 ns (1761103532278.007 ms), delay = 11.508 ms

23:30:26.849 -> 03:30:26.815971

23:30:27.729 -> T1: 295411011000, T2: 1761103827693065900, T3: 1761103827693117500, T4: 295420124000

23:30:27.729 -> Sync: offset = 1761103532278125900 ns (1761103532278.126 ms), delay = 9.061 ms

23:30:27.831 -> 03:30:27.818534

23:30:28.736 -> T1: 296422011000, T2: 1761103828703839500, T3: 1761103828703870900, T4: 296432736000

23:30:28.736 -> Sync: offset = 1761103532277785260 ns (1761103532277.785 ms), delay = 10.694 ms

23:30:28.863 -> 03:30:28.818485

23:30:29.758 -> T1: 297434014000, T2: 1761103829716123400, T3: 1761103829716149800, T4: 297443199000

23:30:29.758 -> Sync: offset = 1761103532277498740 ns (1761103532277.499 ms), delay = 9.159 ms

23:30:29.831 -> 03:30:29.820540

Ln 200, Col 5

Arduino Nano ESP32

on COM6 [not connected]

The above results from the NTP code shows normal delay and offset jitter, consistent with other NTP implementations.

GPS PPS Check  
09:20:41.858 -> Sync: offset=1761311994438.293 ms, smoothed=1761311994439.823 ms, delay=9.474 ms

09:20:41.858 -> Offset Diff: 1584

09:20:41.858 -> 2025-10-24 13:20:41.861013 UTC

09:20:42.089 -> 1000005 , 999951

09:20:42.910 -> NTP: Response timeout

09:20:42.948 -> Sync: offset=1761311994437.480 ms, smoothed=1761311994439.745 ms, delay=7.701 ms

09:20:42.948 -> Offset Diff: 2343

09:20:42.948 -> 2025-10-24 13:20:42.920296 UTC

09:20:43.087 -> 1000007 , 999929

09:20:43.953 -> Sync: offset=1761311994437.582 ms, smoothed=1761311994439.702 ms, delay=8.490 ms

09:20:43.953 -> Offset Diff: 2163

09:20:43.953 -> 2025-10-24 13:20:43.927900 UTC

09:20:44.102 -> 1000007 , 999964

09:20:44.937 -> Sync: offset=1761311994436.826 ms, smoothed=1761311994439.634 ms, delay=11.934 ms

09:20:44.937 -> Offset Diff: 2876

09:20:44.937 -> 2025-10-24 13:20:44.939291 UTC

09:20:45.090 -> 1000006 , 999937

09:20:45.962 -> Sync: offset=1761311994439.557 ms, smoothed=1761311994439.657 ms, delay=12.552 ms

09:20:45.962 -> Offset Diff: 77

09:20:45.962 -> 2025-10-24 13:20:45.950932 UTC

09:20:46.111 -> 1000006 , 1000029

09:20:47.022 -> NTP: Response timeout

09:20:47.064 -> Delay too high: 31121 us

09:20:47.064 -> 2025-10-24 13:20:47.033225 UTC

09:20:47.103 -> 1000007 , 1000008

09:20:48.066 -> Sync: offset=1761311994438.451 ms, smoothed=1761311994439.551 ms, delay=11.466 ms

09:20:48.066 -> Offset Diff: 1206

09:20:48.066 -> 2025-10-24 13:20:48.044739 UTC

09:20:48.105 -> 1000006 , 999900

09:20:49.076 -> Sync: offset=1761311994437.802 ms, smoothed=1761311994439.495 ms, delay=9.490 ms

09:20:49.076 -> Offset Diff: 1749

09:20:49.076 -> 2025-10-24 13:20:49.053719 UTC

09:20:49.076 -> 1000005 , 999949

Conclusion, NTP is waaayyyyy worse than GPS PPS. Just from this quick test its around 50 microseconds off every second, which is substantial drift for my application in the timing gates. Time change smoothing is going to be needed, and more research needs to be done to see how well two devices can sync time. As long as they can achieve millisecond accuracy then this method will be acceptable. Clock speed correction might also be needed, but its looking like NTP is too imprecise to do this kind of calculation.