I^2C Interface to GPS Devices, via the XIAO RP2040 *(Document Version 1.4)*

Robert W Senser, PhD

May 8, 2025

Abstract

In this document we describe our simple I^2C hardware interface to commodity GPS boards that do not support I^2C. The inexpensive XIAO RP2040 board maps I^2C requests to the attached GPS. The XIAO board has two processors; we use one to handle the I^2C requests and the other to handles the GPS serial communications. The intended use of this interface is to allow an existing system, supporting I^2C, to easily communicate with a low-cost GPS.

1 Warning and Disclaimer

This interface was intended to be used in simple hobby-level devices where access to GPS data via I^2C is desired. It is not an industrial-level interface and should not be used where expensive assets are involved. Use at you own risk; we are not responsible for the results obtained with I2CtGPS, you are!

2 Introduction and Origins

This project was an offshoot from our efforts to produce low-cost, hobby-level drone controllers using commercial-off-the-shelf (COTS) hardware. The low-cost GPS boards we have used support a serial interface but not an I^2C interface. The GPS serial interface tended to output long strings of verbose text, which needed to be parsed. Our experience has been that the parsing and processing of this asynchronous, serial data can seriously impact the"architecture" of an existing controller application.

3 Processor Hardware

The Seeed Studio XIAO-RP2040 was chosen because of its low-cost, small form-factor and dual processor support. We used the Arduino IDE, with the XIAO-RP2040 support added, to program the processor. Our C++ code used one

RP2040 processor to communicate with the GPS device and the other to perform I^2C communications. This is a 3.3 volt device; it is not 5 volt tolerant.

4 GPS Boards Used

As of this writing the following inexpensive GPS boards have been used:

1. blox NEO-6M GPS
2. DIYmalls BN-220 GPS
3. BN-180 GPS

These boards support 3.3 volt operations and have serial support.[1]

5 Circuit Layout

Circuit diagram table

XAIO	pin	BN-220	pin
SDA	D4	to client	
SCL	D5	to client	
TX	D6	RX	pin 3
RX	D7	TX	pin 2 (optional)
3.3v	3V3	VCC	pin 4
GND	GND	GND	pin 1
5.0v	5V	not used	

Notes: Common ground to I^2C client not listed. Some thought is needed in powering the GPS device. Does it need 3.3v or 5.0v? The I/O pins on the RP2040 required 3.3v.

6 Software

6.1 Software Files and Location

This software is available for download at URL https://github.com/rwsenser/I2CtGPS. Listed below are the key files in the GIT repository:

1. **GPStester/GPStester.ino**: Sample and test program.
2. **GPStesterUNO/GPStesterUNO.ino**: Sample and test program for UNO.
3. **GPStester/I2Cstruct.h**: Common include file.

[1] Please search the web for the specifications of the hardware mentioned in this document.

4. **I2CtGPS/GPS.h**: Main source code of interface.
5. **I2CtGPS/I2Cstruct.h**: 2nd copy of **I2Cstruct.h**, included to simplify Arduino builds.
6. **I2CtGPS/I2CtGPS1.ino**: Interface driver code.
7. **I2CtGPS.pdf**: This document.

6.2 Software Architecture Summary

The RP2040 has two processors and each is used. The setup() and loop() functions, for the first processor, were used to handle the I^2C processing. The setup1() and loop1() functions, for the second processor, were used to input and parse the serial GPS input. Note that the GPS data was parsed into the **myGPS** class data members. Once all of the needed data from one input sentence was obtained, these data values were copied to the **data** structure. The values in the **data** structure were passed to I^2C via buffer2.[2] This approach was reliable and did not utilize any locking.[3]

6.3 Debugging Hints

1. Testing the GPS inside a building might not work, move the GPS device outside to receive direct signals.
2. macro: The **VERBOSE** macro, defined in **I2CtGPS1.ino**, can be used to have the formatted GPS data serially output, in addition to I^2C.
3. macro: The **GPS_PASSTHRU** macro, defined in **GPS.h**, can be used to serially output the input raw GPS data.
4. suggestion: If the GPS device in use has a serial port then it can be helpful to temporarily connect this port to a PC and observe the raw output. This output should show $GNGGA and/or $GPGGA sentences, when satellites are present.

The two macros should be normally undefined. When either is defined, the XAIO device needs to have a USB device attached. If not attached, the XAIO device might hang.

7 Problems and Noted Improvements

7.1 Problems and Limits

1. Poor status and error reporting.

[2] See the source code for full details.

[3] This approach would likely encounter corruption with the smaller AVR processors where moving **int**s and **float**s requires multiple hardware instructions.

2. Serial communication from XAIO device to GPS device was omitted. The serial output is not wired to the GPS device.[4]

3. Tested as a hobby-grade product. For example, we did not research how this interface behaves if the GPS signals fade over time.

7.2 Possible Improvements

1. Add enhancements and testing to handle known problems and limits.

2. Specifically, this interface could return more data to the using app concerning the accuracy of the GPS data being returned. Currently only the number of satellites is provided.

8 Usage Example

Code snippet from **GPStester.ino**:

```
#include <Wire.h>
#include "I2Cstruct.h"   // provides GPS data structure and I2C address

char buffer[64];
char buffer2[64];
struct I2Cstruct data;

void setup() {
  Wire.begin();        // join i2c bus (address optional for master)
  Serial.begin(9600);  // start serial for output
  Serial.println("\nGPS Reader Running...");
}

void loop() {
  int len = sizeof(data);
  Wire.requestFrom(I2C_ADDRESS,len);   // address from include struct header
   // request len bytes from peripheral device
  int i = 0;
  while (Wire.available()) { // peripheral may send less than requested
    char c = Wire.read(); // receive a byte as character
    buffer[i] = c;
    i++;
  }
  memcpy(&data, buffer, sizeof(data));
  sprintf(buffer2,"%d, %d, %.2f, %f, %f",data.numSats, data.cnt, data.alt,
  data.lat, data.lng);
```

[4]There were sometimes issues with this wired when using 5 volt power to GPS devices. We did not explore this issue further.

```
  Serial.println(buffer2);
  delay(500); // wait half a second
}
```

Sample output:

```
GPS Reader Running...
6, 191240, 5279.53, 39.948238, -112.033966
6, 191255, 5279.53, 39.948238, -112.033966
6, 191280, 5279.20, 39.948242, -112.033966
6, 191295, 5279.20, 39.948242, -112.033966
6, 191320, 5277.56, 39.948242, -112.033951
6, 191335, 5277.56, 39.948242, -112.033951
......
```

Note: The number of satellites will show -9999 before valid GPS readings are present.

9 Usage Pictures

- Figure 1 shows three I^2C sensors and two processors. The XAIO ESP32-C3 processor on the left is connected to an GY-BM 280 sensor, a MPU-9250/6500 sensor and the I2CtGPS sensor, at the bottom. The Pico RP2040 is unattached and was used in a previous test.
- Figure 2 shows an Arduino being used with the I2CtGPS interface. Note the level shifter, on the green board, being used to connect the 5 volt UNO with the 3.3 volt XAIO RP2040, which is positioned below the GPS device.
- Figure 3 shows the first Prototype I2CtGPS top and bottom views.
- Figure 4 shows the side view of the Prototype I2CtGPS interface. Note the foam layer situated between the GPS and XAIO boards.
- Figure 5 shows our second, and final, prototype. The I2CtGPS interface is on the right side of each picture. The XAIO RP2040 on the left is functioning as the I^2C client. The XAIO on the left powers the I2CtGPS interface on the right. The other two I^2C sensors are omitted from this prototype.
- Figure 6 shows our fearless project manager and tester.

10 Conclusions

This approach to GPS access via I^2C has worked well for hobby use.

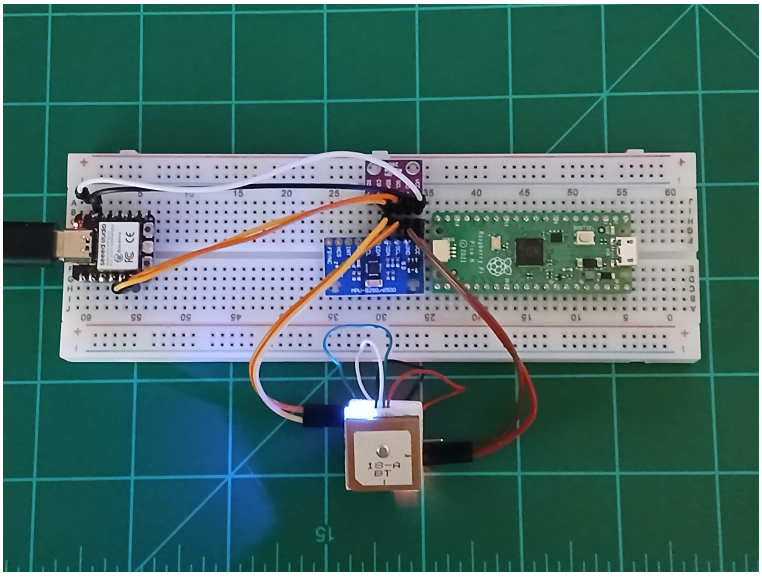

Figure 1: Top View Test Setup: shows several I^2C sensor devices and several possible clients.

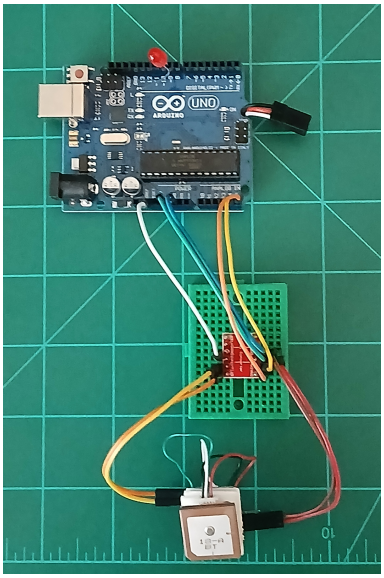

Figure 2: UNO Top View with Level Shifter

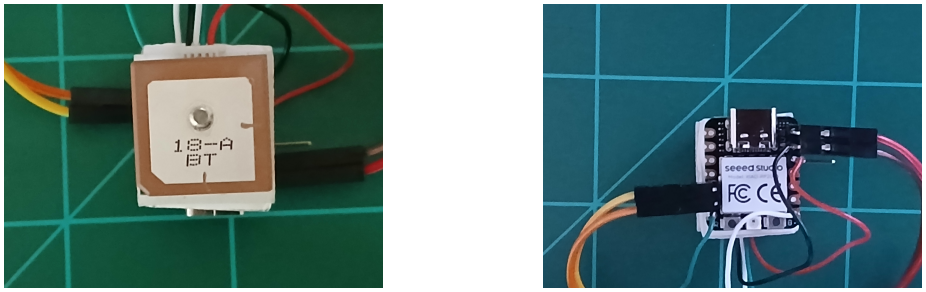

Figure 3: First Proto Top and Bottom Views

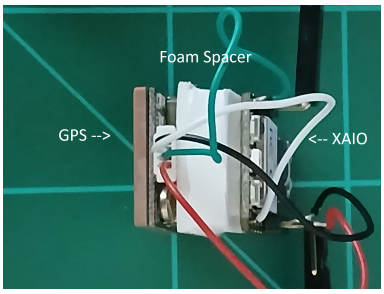

Figure 4: First Proto Side View

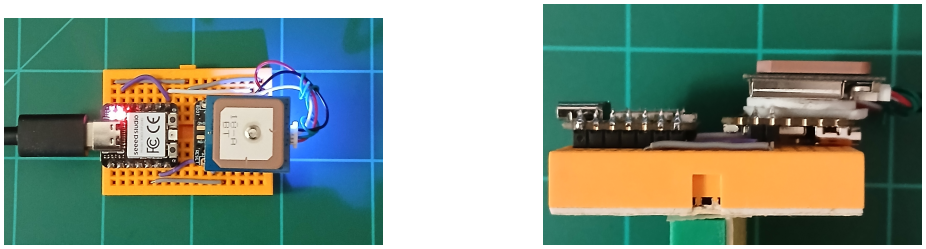

Figure 5: Final Proto, Top and Side Views

Figure 6: Our AI-generated “fearless” Project Manager