Sending Adafruit GPS receiver data wirelessly to an Arduino Mega 16 MHz microcontroller using the XBee Series 3 transmitter-receiver system

DAVID BUITRAGO

YORK COLLEGE AND BNL SULI SUMMER INTERNSHIP PROGRAM

JULY 15, 2019

Our application

We are building a data acquisition circuit (DAQ) for cosmic ray muon detectors made from plastic scintillators and photomultiplier tubes. Our DAQ will consist of a two-channel DAQ front end circuit (amplifiers, discriminators, coincidence logic, and signal peak detectors) connected to an Arduino microcontroller which digitizes the DAQ FE data and time stamps cosmic ray signals with GPS UTC time.

The GPS antenna signal is sent to a GPS receiver which communicates to our Arduino; the antenna has to be near a window to see satellites however at some detector sites the Arduino will be located far from the window and wireless is desired to avoid having to run long cables which attenuate the signal, and which building rules sometimes do not allow.

We want to compare the GPS timing resolutions for the wired and wireless methods; to do this we measured the number of Arduino clock cycles between successive GPS 1 PPS pulses.

We have tried these two types of wired GPS connections:

Wired method 1) a short ~5m antenna cable to the receiver, and a long ~100 ft. CAT6 network cable from the receiver to the Arduino; this ensures lower attenuation on the antenna analog signal.

Wired method 2) a longer ~100 ft. antenna cable to the receiver, and we plug the receiver directly into the Arduino. We connect them this way when we don't want to leave the receiver by the window. The problem with this method is the longer antenna cable attenuates the signal.

Equipment

Arduino Mega 2560

Adafruit RF GPS Antenna 960, 1.575 GHz active antenna with 28 dB gain,

Adafruit GPS Breakout 746, 1.575 GHz, 66 Ch with 10 Hz updates

Adafruit GPS Logger Shield 1272 for Arduino, 66 Ch with 10 Hz updates

RG174 coaxial cable

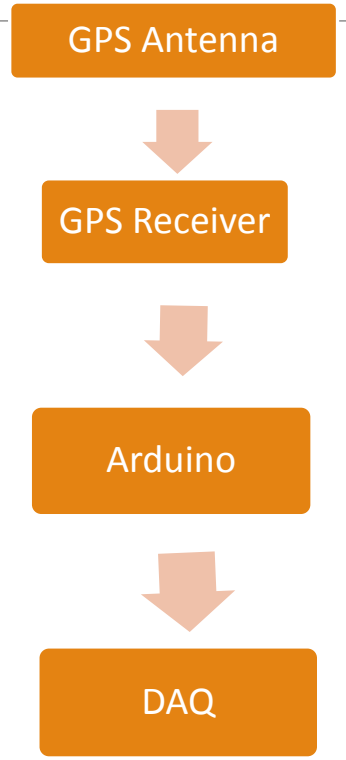
XBee Series 3

The first part of my project was understanding Arduinos and the code a previous student (Junjie) had already created to operate our Arduino for data acquisition.

Arduinos are powerful microcontrollers that can grab and print serial data, send and receive digital data and convert analog data to digital data.

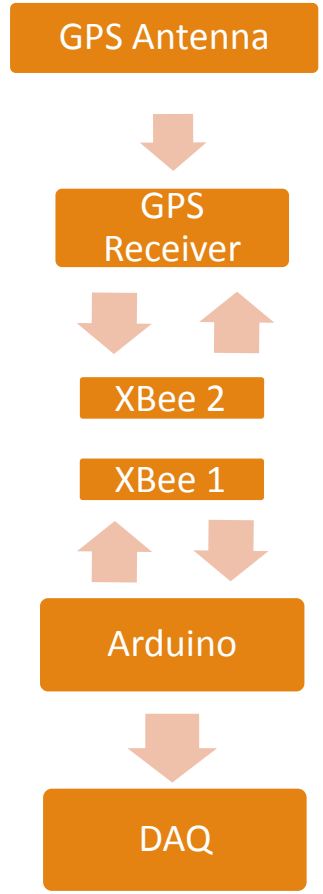
Our DAQ board will consist of a two channel DAQ front end board connected to the Arduino microcontroller and GPS.

Wired vs Wireless:



Wired :

For the wired setup the receiver can be close to the antenna or close to the Arduino. In either case a long cable is required which can be inconvenient in places that require additional permissions to run cables.



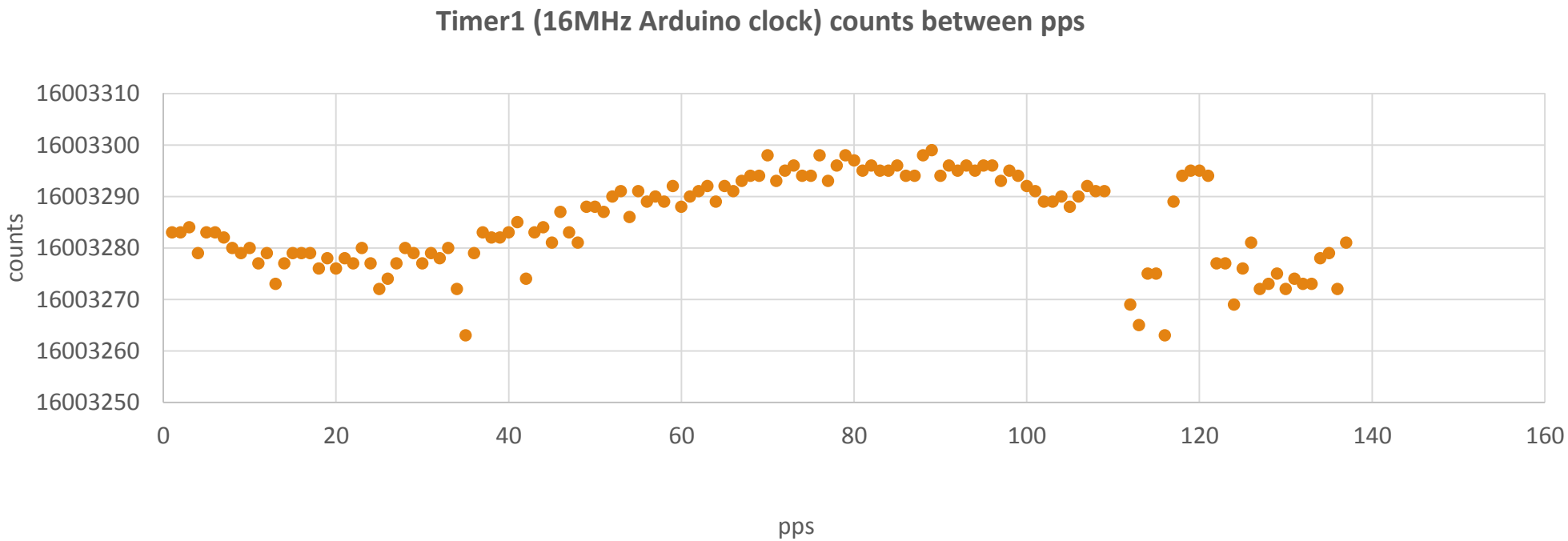
Wireless:

For the wireless setup the receiver is near the antenna and a long cable is not required, thus no cable attenuation to the signal.

Results for wired method 1:

Here are Junjie's previous measurements of the number of clock cycles between successive PPS pulses for a wired connection; the wire from antenna to receiver was 5m, and the network cable from receiver to Arduino was 100 ft. The results show oscillator drift or jitter of about of +/- 20 clock cycles which corresponds to +/- 1.25 microsec resolution

+/- 1.25 ms resolution when using the wired GPS connection
(a short cable ran from antenna to receiver and a long network cable from receiver to Arduino)



2 mins of data collection
about +/-20 count spread (0.0002% variation)

Results from wired method 2

For this we used 100 ft of RG174 coax cable running from the GPS antenna to the GPS receiver shield and plugged the receiver directly into the Arduino. The receiver did not receive the antenna signal; so we measured the cable attenuation by injecting a square pulse down the cable and measuring its power coming out the other end with an oscilloscope terminated in 50 Ohms.

$P = V^2/R$

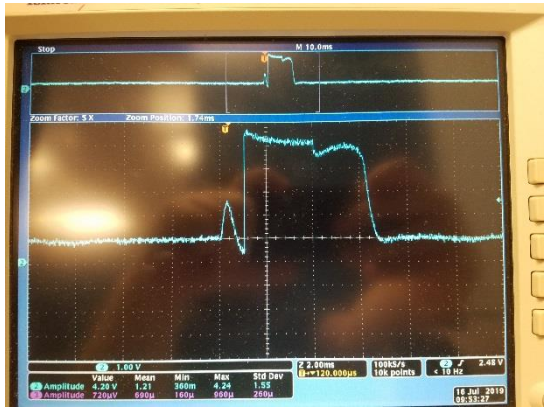
- $P_{in} = (5V)^2/50$ Ohms = 0.5 Watts
- $P_{out} = (0.4V)^2/50$ Ohms = 0.0032 Watts

- Cable attenuation = $10*\log(P_{in}/P_{out})$ = 22 dB

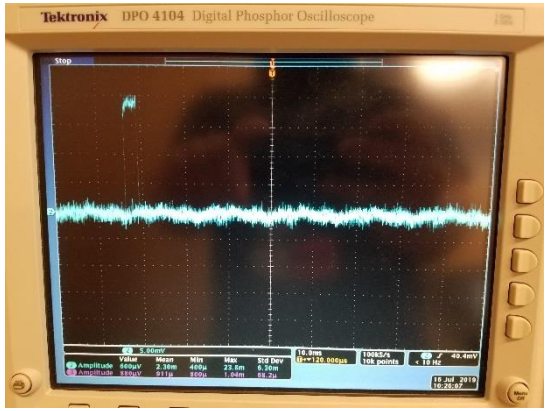
- The spec sheet for RG174 cable says at 1 GHz it has 32 dB per 100 ft.

The GPS antenna specs say it has 28 dB of gain; we are not sure why the receiver is not seeing the antenna signal. We will remeasure the cable attenuation but this time for a 1.575 GHz signal which is the frequency of the GPS antenna signal.

At a different detector site we used 100 ft. of RG58 instead of RG174 and the GPS receives the antenna signal.



Input Signal



Output Signal

Results from wireless method:

The required specifications for the wireless controller are:

- Have a baud rate of 9600 (matching our GPS receiver).
- Have the correct data transfer protocol to handle fast data transfer.
- Have available resource materials.

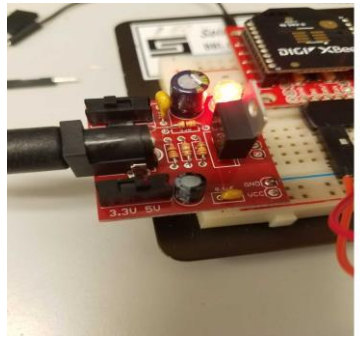
The XBee fits all of these requirements; other wireless technologies exist but there was not much information on their use with an Arduino; XBees seem to be the standard when it comes to wireless communication with Arduino.

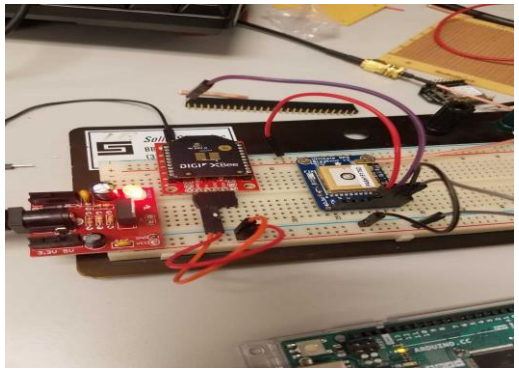
XBee series 3 has the added benefit of itself being a microcontroller capable of sending messages to a serial port.

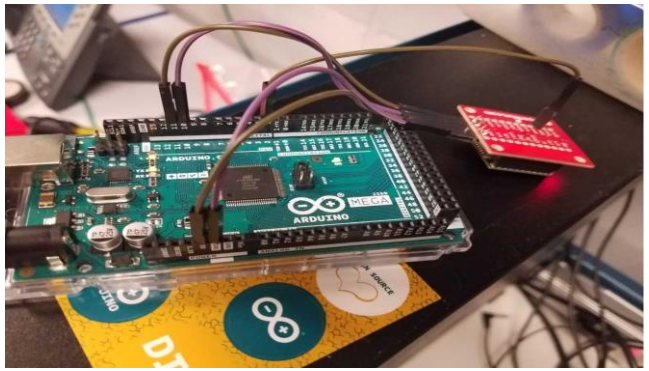
Wireless Setup

Once the equipment was purchased the next step was to solder and put together the circuitry ,and begin transferring GPS data.

The multipoint 802.15.4 WIFI protocol is used; Digimesh and ZigBee protocols were considered but they take longer time to transfer data.







Goals for GPS data

With the setup now complete the next step is to wirelessly transfer from GPS to Arduino the 1 PPS signal and the NMEA serial data.

The goals for the GPS data:

1. Obtain the 1PPS signal for the start of each second of UTC time. This was achieved by configuring the XBee to forward all incoming signals from the 1PPS pin on the adafruit receiver to the Arduino.

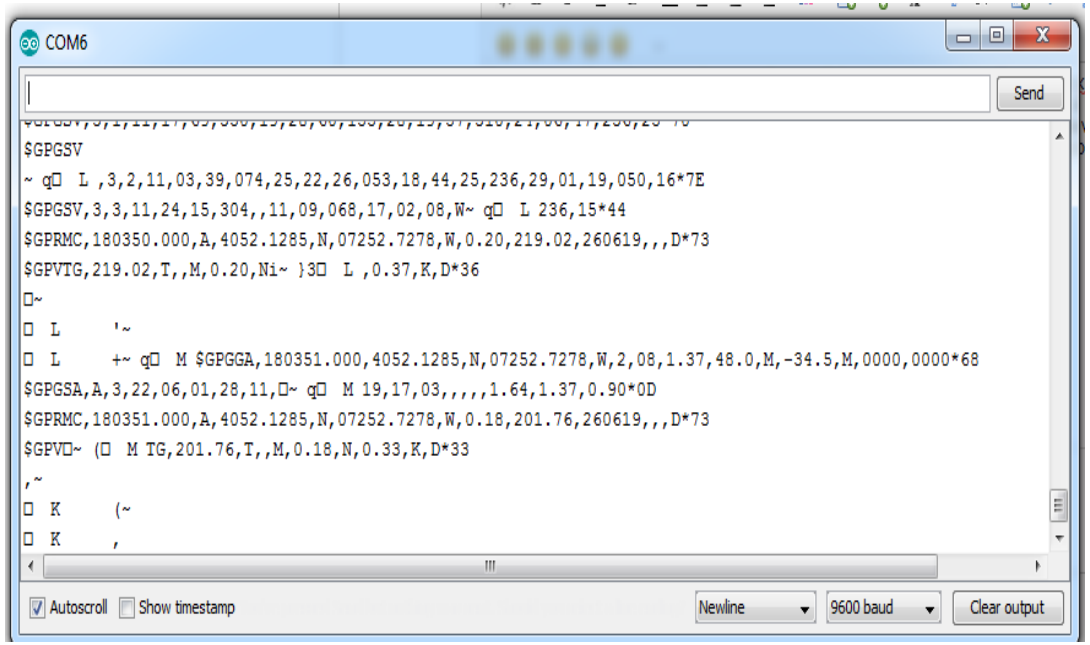
2. Measure any variation in the number of Arduino clock cycles between the receipt of successive PPS pulses to quantify timing uncertainties

3. Print and save NMEA sentence data (such as date, time, longitude, latitude, and number of satellites). This was achieved by connecting to the RX TX pins on the adafruit receiver and figuring out how to print the data to the serial monitor.

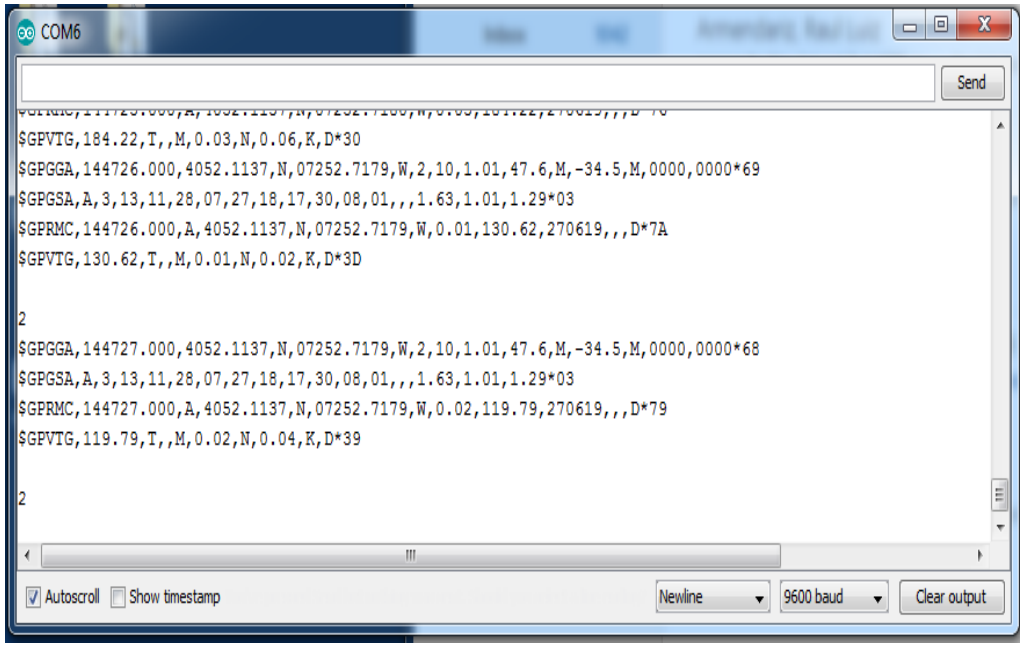
GPS Data Examples

At first the NMEA data got corrupted when we wirelessly transferred it but this has been fixed.

Corrupted Data



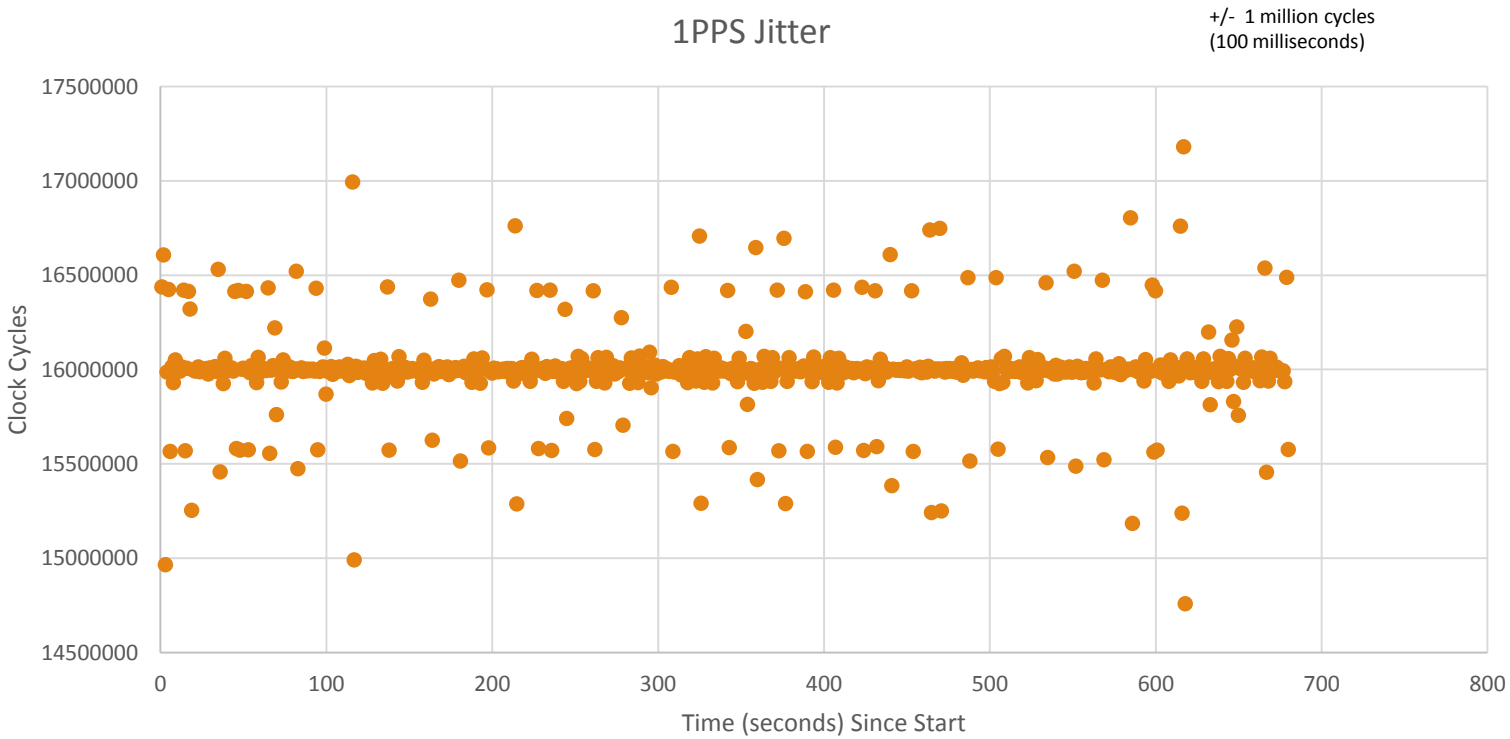
Fixed Data



1PPS Jitter

The next step was to observe the 1PPS jitter; code was written for the Arduino to record the number of clock cycles between successive PPS pulses. The results show inconsistent jitter of about of +/- 10^6 clock cycles or +/- 62.5 ms, which seems to increase over time. However the majority of time the number of cycles land around the expected 16 million. We are not sure what the cause of the jitter is, it may be related in part to the GPS module not being configured.

+/- 62.5 ms resolution when using the XBee wireless for GPS



Remaining Goals For Summer

Finish programming the Arduino script to parse GPS NMEA data (eliminate copies of repeating sentences).

Find a way to program the XBees to pass commands from the Arduino to the GPS so that we can select which NMEA data we want and how often we want to receive GPS time information.

Find an ideal enclosure for the Xbee and GPS wireless electronics.

If there is time look more carefully at the 1 PPS jitter.