Developing a wireless GPS data collection system for cosmic ray timing (summer 2019)

David Buitrago, Physics, York College, Jamaica, NY, 11451

Raul Armendariz Ph.D., Physics, Queensborough Community College, Bayside, NY, 11364

David Jaffe, Physics, Brookhaven National Laboratory, Upton, NY, 11973

Abstract

The Quarknet experiment aims to measure and study muons that reach the earth's surface from incoming cosmic rays when they collide with atoms in the atmosphere (mostly nitrogen) and create a "shower" of particles, including muons. The energy of these cosmic rays can be determined by measuring the area of the shower created by the collision. Through a collective effort from cosmic ray detectors operated at various schools across the country, cosmic ray shower data is being gathered to investigate high energy cosmic rays. Precise time and position information from each detector are needed to measure the cosmic ray showers. One inconveniences of our current system is the long cable between the GPS receiver and our data acquisition (DAQ) board to obtain timing data. One of the solutions to this problem and the main goal of this summer project is to demonstrate the capability of wireless data communication to replace the long cable. We wish to measure the accuracy of wireless to wired GPS data and determine if our hardware is also limiting the speed at which we receive data.

I. Introduction

As cosmic rays hit our atmosphere they collide creating elementary particles, the main topic of our research is one of these particles, the muon. Ordinarily these particles would never reach Earth's surface and be detected but due to special relativity they last much longer than their decay rate would suggest. Muons are being closely studied all over the world for various topics but the focus of our project is finding high energy muons by searching for muon showers. Our project depends on using a DAQ board to determine if an event is a coincidence, as their name suggests, coincidences are when two or more detectors see a particle at the same time, while this does not always mean that the event is a muon, the tolerance on our DAQ is around 100ns and so we can be reasonably sure we are not seeing noise from our detectors. Muon showers can be measured by creating an array of detectors and waiting for coincidences to occur, after which if they meet our timing criteria they are considered an event and become part of our data.

One of the most important aspects of our project is the placement of the hardware that we are using to measure cosmic ray muons. To accurately time the events we must use a GPS to determine our longitude and latitude, time, and most importantly to measure the one pulse per second sent out by our receiver to coordinate the timing between our detectors. This connection to the GPS requires two cables to be run from our receiver to

the external antenna we have, which must be placed near a window to accurately detect satellites, and to our DAQ. Because we want to host these detector arrays at various sites that may all have varying restrictions on running cables paired with the fact that the antenna may not necessarily be in the same room as the receiver and DAQ board we want a form of wireless communication between our devices that will still allow us to get GPS data.

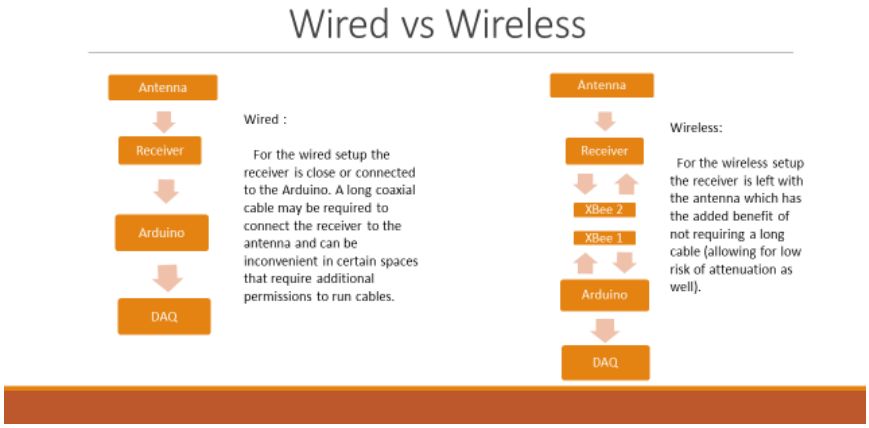

Figure 1: Wireless vs Wired

II. Methods

A. Wireless Setup

First, before dealing with the wireless data we must connect an Arduino to our DAQ board that will process the data we send it and help with timing the muons that are caught by our detectors. This Arduino must also be connected to our wireless device to receive GPS data in this setup and a script must be written to parse the data and timestamp our coincidences appropriately. While code had previously been written in C++ that automatically parses the serial data, because the format has changed to wireless we must adapt the script to parse the data manually.

One form of wireless communication and the main focus of this research is a microcontroller called X-Bee. They are capable of various forms of wireless protocol, from Bluetooth to 802.15.4, and also have onboard memory to flash scripts onto using micro python. The latency of 802.15.4 is not consistent unfortunately, it is affected by distance, number of nodes, size of messages, physical obstructions (walls, metal). The range of the X-bees also depends on line of sight and it's recommended not to place glass or concrete between two X-Bees because the effective range gets diminished very quickly. Of the available protocols however 802.15.4 is the best at communicating with a single point, the others handle multiple points faster.

This allows for involved communication between our devices when sending GPS data. Our setup was connected as follows: an Arduino to a receiver X-Bee and an X-Bee connected to our receiver/antenna. The following pictures show the wireless local and remote setup:

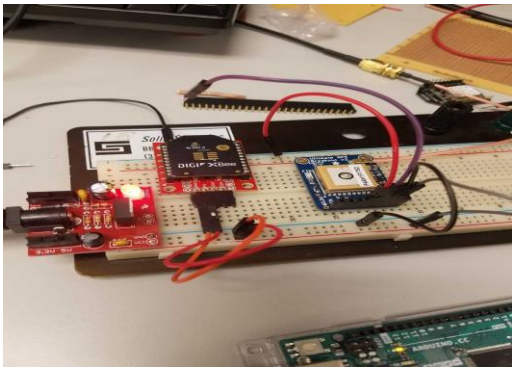

Figure 2: Transmit X-Bee

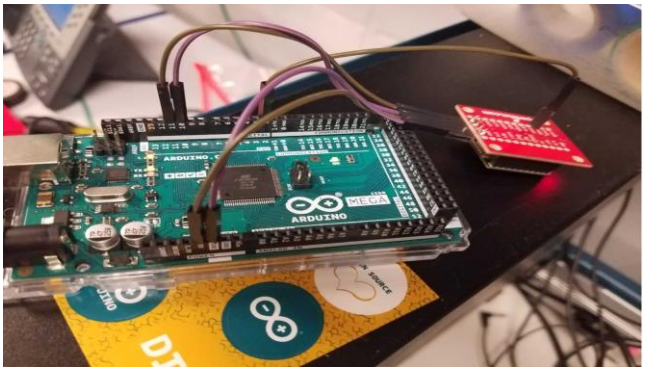

Figure 3: Receive X-Bee

B. One Pulse Per Second

The one pulse per second is a signal generated in our receiver that can be used to maintain synchronization with UTC, the accuracy of this signal is around 100ns and it goes high(5V) at the beginning of every second. This pulse plays an important role in our event timings because we may need up to microsecond precision to view events that are happening multiple times a second. We use this pulse to measure how much time has elapsed between events and the start of a new second.

C. X-Bee

For the wireless communicators that must be used to get data from our GPS receiver to our Arduino there were a few options. Amongst the required specifications of the wireless controller it must; have a baud rate of 9600 (matching our GPS receiver), have the correct data transfer protocol to handle fast data transfer. The X-Bee fits these requirements and while other wireless communicators were considered there was not much information on them anywhere, X-Bees seem to be the standard when it comes to wireless communication.

D. NMEA Sentence

NMEA Sentences provides GPS information from our receiver constantly, in our case at a rate of 1Hz. They are produced by our receiver but populated by satellites feeding relevant information to our setup, this includes number of satellites seen, general coordinates, UTC time, etc..

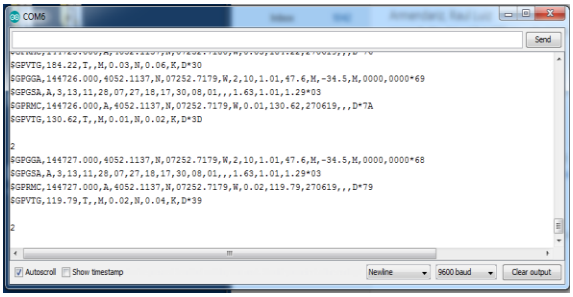

Figure 4: Raw NMEA Data

III. Results

While the NMEA data and the one pulse per second are both sent by the receiver the way they are sent and the method to output them is different. As such the results will be split between these two main topics.

A. 1PPS

The results for the 1PPS were interesting, while the X-Bee can handle triggering, the process overloads the serial buffer and causes corruption of the data.

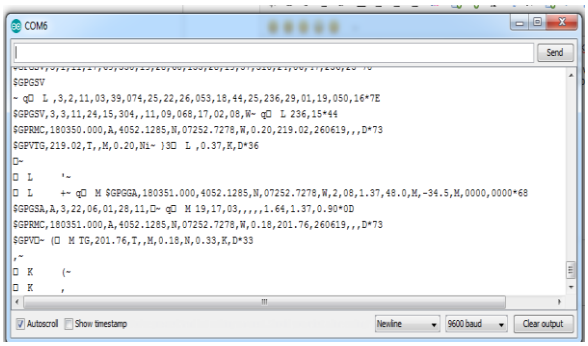

Figure 5: Corrupted Data

Due to this corruption of data the Arduino was chosen to handle both parsing and triggering. A script was written in C++ to calculate the difference in clock cycles between two pulses to measure if our wireless setup received and timed the signal at the same rate as our wired setup. We did this by measuring the number of overflows that accumulated as our counter increased and multiplying that by the maximum number of clock cycles before an overflow (256) then adding the clock cycles that elapsed between the start of the 16MHz clock and when we receive the 1PPS.

$$(\#\ of\ overflows) * 256 + (elapsed\ clock\ cycles)$$

The graphs below illustrate the differences in our wired vs wireless setup jitter, the wireless setup had a very high amount of jitter (100ms) which is outside the allowed tolerance of our project.

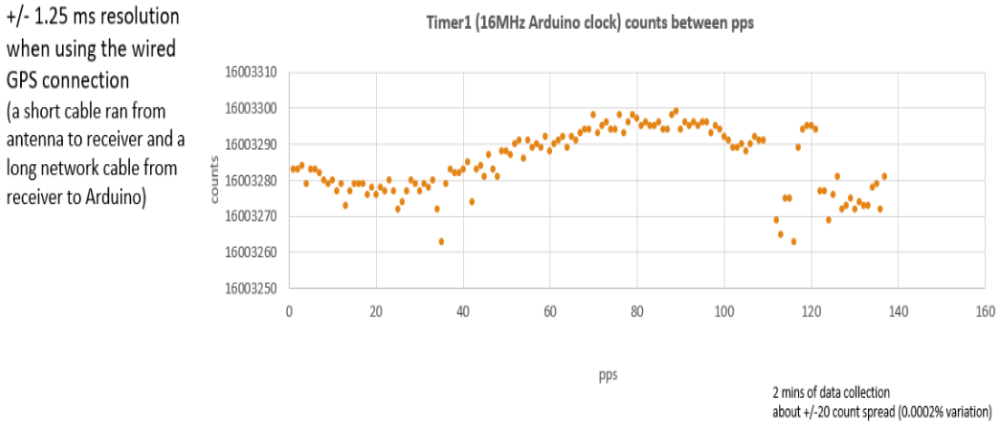

Figure 6: Wired PPS Jitter

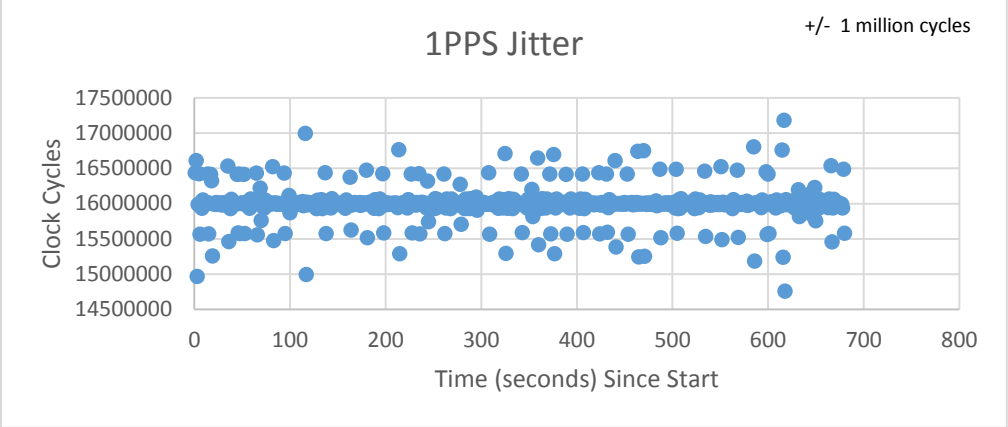

Figure 7: Wireless PPS Jitter

B. NMEA data

The NMEA data required a different strategy, for the wired setup we had a way of communicating directly with the GPS with the help of libraries written for this specific purpose. For our wireless connection however because the Arduino and the GPS are now separated by X-Bees we cannot configure the receiver and must parse the serial data manually. Of the available NMEA sentences we require only the GPRMC sentence as it holds all of the useful information for timing events, however the script was written to parse any sentence necessary or multiple ones. The figure below shows both the 1PPS clock cycle difference and parsed NMEA data.

```
GPRMC,160930.000,A,4052.1312,N,07252.7247,W,0.45,89.66,230719,,,D

16000138.00

GPRMC,160931.000,A,4052.1311,N,07252.7246,W,0.44,52.27,230719,,,D

16001616.00



GPRMC,160932.000,A,4052.1310,N,07252.7246,W,0.36,29.69,230719,,,D
15931655.00


GPRMC,160933.000,A,4052.1311,N,07252.7245,W,0.30,36.34,230719,,,D

16054857.00

GPRMC,160934.000,A,4052.1312,N,07252.7245,W,0.44,48.16,230719,,,D

16000033.00

GPRMC,160935.000,A,4052.1312,N,07252.7243,W,0.19,29.11,230719,,,D

16006212.00

GPRMC,160936.000,A,4052.1313,N,07252.7243,W,0.14,5.73,230719,,,D

15988905.00
```

Figure 8: Complete Data Output

IV. Conclusions

While unsure at the beginning of this project we have shown conclusively that you can add wireless communication between a GPS and Arduino and receive real time data. However, the jitter in the 1PPS shows

that while a wireless setup can provide convenience when having to setup the detectors for data taking the accuracy of these readings is above the maximum allowed inaccuracy to time events.

For the serial NMEA sentences they are now properly working and being sent on time, the tolerance for these sentences is much higher, as long as they come in after the first PPS and before the second the can define the time up to the most recent second. In the future, if the rate at which these sentences are sent increases, the jitter will once again become a problem for the sentences as well.

V. Acknowledgements

- This project was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Science Undergraduate Laboratory Internships Program (SULI).