CarSystem GPS How-To Guide

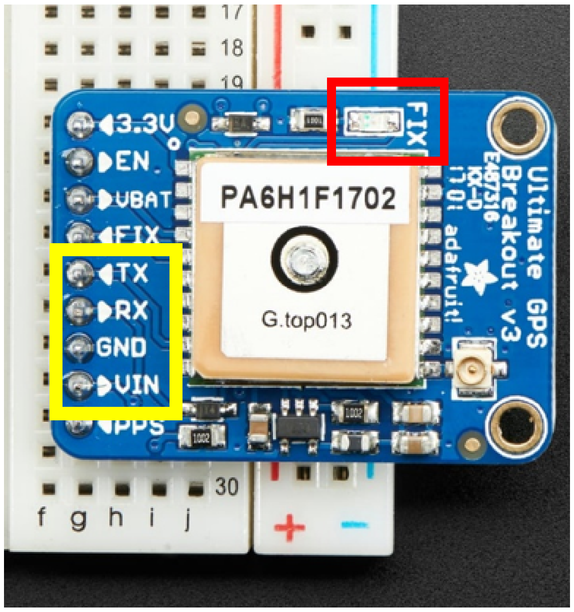
By:
Divya Soneji, Chirag Gupta, Peeyush Sharma & Paramdeep Manhas

Introduction:

This document serves as a comprehensive how-to guide for setting up the GPS component of the CarSystem project and reading data from it inside a C Program. It provides step-by-step instructions on how to assemble the GPS component, configure it properly, as well as compile and run the project. It also includes some helpful hints for making sense of the raw data. Our group's primary use case was to determine location via latitude and longitude as well as determine the speed of our Beaglebone since we were attempting to emulate a car system. We have included a gps.c code which may be used as a reference file in case the reader wants to see our full implementation (however all important snippets of code are directly in this pdf).

The GPS we used is an external hardware item that our team bought ourselves and is made by Adafruit: [Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates [PA1616S] : ID 746 : $29.95 : Adafruit Industries, Unique & fun DIY electronics and kits]

Assembling GPS Hardware:



- ➢ Begin by connecting the GPS module to any row on your breadboard.
- ➢ Next, you will need **four jumper wires** to connect the four pins of the GPS module (as highlighted in yellow) to your Beaglebone.
- ➢ Pin numbers below are all for **P9**
- ➢ **Ground**: Connect the GND (ground) pin of the GPS module to Pin 1 on your Beaglebone.
- ➢ **Communication**: Connect the TX (transmit) pin of the GPS module to Pin 22 on your Beaglebone, which is the receive pin. Connect the RX (receive) pin of the GPS module to Pin 21 on your Beaglebone, which is the

transmit pin. *Note: It's very important you connect the TX to RX and not to TX (it must be **inverted** in that sense).*

- ➢ **Power**: Connect VIN to Pin 7 of the Beaglebone to establish a power connection.
- ➢ After you have connected all of the pins, you should hear a beeping sound indicating that the GPS has successfully powered up.
- ➢ If you see a flashing red light accompanied by a beeping sound on the LED (highlighted in red), it means your GPS module is not receiving a signal. This can occur when you are indoors because the roof will block the satellite connection. In that case, you will likely need to go outside to receive a signal on your GPS module. You will know your GPS has been successfully wired up when you see the red dot and hear the beeping sound. This should happen simply on successful power for GPS and nothing being run.

Configuring Your System:

This section walks you through step-by-step instructions on how to properly wire the different hardware components together.

NOTE: The terminal commands that start with a `$(bbg)` *indicate that the commands need to be run in the target, whereas those that start with a* `$(host)` *indicate that they need to be run in the host.*

- ➢ To configure the Beaglebone for serial communication with the GPS module, you need to set pins 21 (transmit) and 22 (receive) of your Beaglebone as UART. This can be achieved by running the following commands on your target:

```
$(bbg) config-pin P_21 uart
$(bbg) config-pin P_22 uart
```

How to Compile and Run And Get Some Raw Data:

First, it is important to define some base rates at the top of your .c file that will become important to communicate with the GPS later on.

```
#include <fcntl.h>
#include <termios.h>
#define SERIAL_PORT "/dev/ttyS2"
#define BAUDRATE B9600
#define CRTSCTS 020000000000 /* flow control */
```

Here the serial port refers to where you are connecting to the GPS, in the case of our Beaglebones, it should be /dev/ttyS2. Note: If you're using Audiono or a different version of Beaglebone it will likely be different (this gave us trouble initially). The baud rate specifies the number of bits transmitted per second. The CRTSCTS is a constant used in programming for Linux systems and specifies the use of hardware flow control for serial communication. It is wise to leave the BAUDRATE and CRTSCTS unchanged unless needed to be changed to be compatible with your system. We have included those two libraries as they will be necessary later on for connecting with the serial port.

```
int fd;
struct termios options;
char buf[255];
// Open serial port
fd = open(SERIAL_PORT, O_RDWR | O_NOCTTY | O_NDELAY);
if (fd == -1)
{
   printf("Error: Unable to open serial port\n");
   // return -1;
   exit(-1);
}
```

The code above opens a serial port which is important since we gps uses a serial port connection and we need to open that serial port in order to read the gps data (the path to which we defined earlier in #define SERIAL_PORT).

Next let's configure the serial port for gps usage using the following:

```
    // Configuring serial port
    tcgetattr(fd, &options);
    cfsetispeed(&options, BAUDRATE);
    cfsetospeed(&options, BAUDRATE);
    options.c_cflag |= (CLOCAL | CREAD);
    options.c_cflag &= ~PARENB;
    options.c_cflag &= ~CSTOPB;
    options.c_cflag &= ~CSIZE;
    options.c_cflag |= CS8;
    options.c_cflag &= ~CRTSCTS;
    options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
    options.c_oflag &= ~OPOST;
    options.c_cc[VMIN] = 5;
    options.c_cc[VTIME] = 2;
    tcsetattr(fd, TCSANOW, &options);
```

These options check the existing configurations of the serial port and then make changes to make it compatible with the gps including doing things like setting the input and out BAUDRATE, using the 'cfsetispeed' and 'cfsetospeed' functions. It also plays around with some bits configurations for the serial connection and importantly the options.c_cc[VMIN] = 5; and options.c_cc[VTIME] = 2 sets it so that the read function should only read once 5 characters or 2 tenths of a second has passed (which is very quick, you may wish to change this depending on your project requirements).

Finally let's start reading into our buffer some data! Here is simple code to do that:

```
     while (1) {
          int bytesRead = read(fd, buf, sizeof(buf));
     }
```

That's it for reading the data. You can print out the data simply by printing out the char buffer we defined earlier (known as buf)! One tip we would recommend is to have a **busy-wait or some other mechanism** inside the while loop to not read any data if zero bytes are returned, as that data would be useless and might result in blanks. Here is what one iteration of the buffer containing the data might look like:

```
$GPGGA,203754.210,4911.3261,N,12250.9984,W,1,04,1.89,167.7,M,-16.7,M,,*57

$GPGSA,A,3,06,19,17,24,,,,,,,,,2.13,1.89,0.98*0D

$GPRMC,203754.210,A,4911.3261,N,12250.9984,W,2.84,154.47,230323,,,A*75
```

Brief intro to working with the data:

This guide has covered its main purpose which is setting up the GPS module and reading in data from it via a C Program but this section briefly describes how we made sense of the data to be used. Our group used 3 main data points: The Latitude, The Longitude, and the Speed; getting them is described below.

```
1) GPGGA line contains the lat/long data. While the GPRMC
   line contains the velocity data
2) Split both of the lines by commas
```

3) Take the 3rd split + 4th split of GPGGA to get the latitude. The 3rd number is the latitude (but you will need to write your own conversion script to convert it to the commonly known latitude) while the 4th number represents the direction for that latitude. Identical for longitude except for the 5th and 6th split. Getting it in the format to be inputted into google maps will likely take some work. Hint: A particular direction may represent negative values, the gps itself will always give positive values
4) The speed in the GPRMC line is the 7th split.

Troubleshooting Guide:

- **No red light on Power on?** Check your wiring to ensure ground and power are connected properly. Note the tx and rx wiring does not matter to power on.
- **Red light shining but no data gotten from GPS?** Check your RX and TX and ensure the RX wire connects to the TX pin and vice versa! Also check that you have correctly configured the pins to **uart** for your c program to work!
- **GPS not getting signal?** Ensure you are outside and GIVE it some time (like a minute or so), walk around with it. It should eventually get a signal. Once a signal has been received, it should not stutter. Remember to check the GPS signal is working, no beeping sound should occur.
- **Latitude and Longitude not correctly pointing to Google Maps?** Ensure that your conversion function properly gets the values and that you have the right direction set to negative. Finally, keep in mind that latitude and longitude by nature may not be able to pinpoint your exact location. It might showcase a nearby location, this is okay (like for example if you're in Surrey, it might showcase Langley).
- **Data showing lots of blanks?:** Indicates that the GPS does not have a signal or a very weak signal. Look out for occasional beeps. It's worth **noting** that even a no signal will still print data if read from, it will just be mostly blank.

THE END