Grove GPS (SIM28) UART Guide for BeagleBone Green

Guide is tested on:

- BeagleBone Green: **Debian 11.5**
- Host OS: **Debian 11.6**
- Grove GPS: **SIM28 Version (https://www.seeedstudio.com/Grove-GPS-Module.html)**

This document guides the user through:

1. Introducing Grove connectors and the UART communication protocol
2. Initializing the Grove UART port on the BeagleBone Green
3. Interfacing the Grove GPS through Linux's device node to get raw NMEA data
4. Using C code to interface with the Grove GPS module

1. Grove Connectors and UART Protocol

Grove connectors by Seeed Studio (the same company behind the BeagleBone devices) are a connector prototyping system which works as a plug-and-play solution to connect various electronics through a multitude of communication protocols. Grove connectors support 4 kinds of interfaces: Digital, Analog, UART, and I2C. The BeagleBone Green Includes 2 Grove connectors (I2C and UART). For the purposes of the Grove GPS module, we will be using the UART interface.

UART (Universal Asynchronous Receiver/Transmitter) is a simple serial communication protocol to exchange serial data between devices. UART communication consists of a Receive (Rx) channel and a Transmit (Tx) channel. Because UART communication has separate channels for receiving and transmitting, UART allows for full duplex communication (both parties can transmit data at the same time). UART also allows for flow control in the form of RTS (Request to Send) and CTS (Clear to Send) channels but both these channels are not supported on the Grove UART interfaces.

2. Enabling Grove UART port on the BeagleBone Green

The BeagleBone Green's UART Grove Interface is connected to **UART2** which is interfaced by the **P9_21** (UART2_TXD) and **P9_22**[1] (UART2_RXD) pins on the board. In order to enable UART2 on the board, the device tree has to be loaded so Linux knows how to handle the UART connection.

2.1 Loading the UART2 device tree

1. Ensure the UART2 device tree exists on the device
    a. Run the command:
    `(bbg)$ ls -l /lib/firmware/*UART*`
    b. Ensure that the device tree blob `/lib/firmware/BB-UART2-00A0.dtbo` is found
2. Load the device tree in `/boot/uEnv.txt`
    a. Back up the uEnv file before proceeding
    `(bbg)$ sudo cp /boot/uEnv.txt /boot/uEnvBackup.txt`
    b. Edit `/boot/uEnv.txt`
    `(bbg)$ sudo nano /boot/uEnv.txt`
    c. Find the section titled:
    `###Additional custom capes`
    d. Load the .dtbo file onto any of the overlays
    ```
    ###Additional custom capes
    uboot_overlay_addr4=/lib/firmware/BB-UART2-00A0.dtbo
    ```
    e. Reboot the board

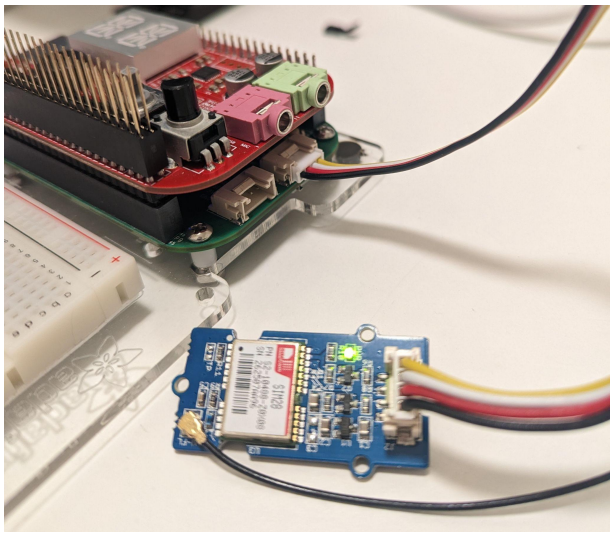
[1] The P9_22 pin is also used by the Zen Cape's Buzzer as a PWM pin interface, so plugging in the Grove GPS module to the BeagleBone Green while the Zen Cape is connected will result in the buzzer beeping. To turn off the buzzer, you can remove the jumper located on the side of the buzzer on the Zen Cape.

3. Connecting the Grove GPS Module to the BeagleBone Green

Once the device tree is loaded, Linux should know how to operate the UART2 port. Now, we can connect the GPS module to receive GPS data. GPS data is transmitted in the form of NMEA (National Marine Electronics Association) sentences. These sentences include important information, such as how many satellites are in view and latitude and longitude of the device.

3.1 Connect and interface with the GPS module

1. Connect the Grove GPS Module to the Grove UART connector on the BeagleBone Green using a Grove Connector Cable. An example is shown below.



   a. Note: Ensure that the device is connected on the UART Grove port, **not** the I2C Grove port.
      i. An easy way to check if the module is connected on the correct port is to listen to the Zen Cape's buzzer. Because UART2 and the PWM interface the buzzer uses is controlled through the same pin, the buzzer should ring if the module is connected to the UART Grove port
2. Print the output of the GPS receiver through the Linux device node
   a. Because UART is a serial communication protocol, Linux can communicate with the GPS receiver through Linux's serial device node.
   b. UART2's device node is found in `/dev/ttyS2`
   c. To see the GPS module's output, run the command:

```
(bbg)$ cat /dev/ttyS2
$GPGGA,010755.800,,,,,0,0,,,M,,M,,*46
$GPGSA,A,1,,,,,,,,,,,,,*1E
$GPGSV,1,1,00*79
$GPRMC,010755.800,V,,,,,0.00,0.00,060180,,,N*4C
```

d. The output of the GPS module is formatted in NMEA sentences. For more details about the NMEA sentence format and how to parse them, refer to the SIM28 NMEA Message Specification[2]

4. Interface with the UART port through C code

Interfacing with the UART port with C code can be done using the `<termios.h>` library, which allows control of asynchronous communication ports. Because the UART communication protocol is an asynchronous serial protocol, we can interface with the UART port through a serial connection.

4.1 Configure a serial connection

The function below shows an example on how to configure the serial port for the Grove GPS module. The function is adapted from Geoffrey Hunter's article[3], where detailed explanations for each configuration can be found:

```
#include <termios.h> // Contains POSIX terminal control definitions

int serialPort; // Contains the file descriptor for the serial port

// Function adapted from Geoffrey Hunter's article:
https://blog.mbedded.ninja/programming/operating-systems/linux/linux-serial-ports-using-c
-cpp/
// Configures the file descriptor fd for serial port read/write
// Returns 1 on error and 0 on success
int configureSerial() {
    struct termios tty;

    // Read in existing settings, and handle any error
    // NOTE: This is important! POSIX states that the struct passed to tcsetattr()
    // must have been initialized with a call to tcgetattr() overwise behaviour
    // is undefined
    if(tcgetattr(serialPort, &tty) != 0) {
        printf("Error %i from tcgetattr: %s\n", errno, strerror(errno));
    }

    tty.c_cflag &= ~PARENB; // Clear parity bit, disabling parity (most common)
    tty.c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit used in communication
(most common)
```

[2] SIM28/68R/68V NMEA Messages Specification
[3] Linux Serial Ports Using C/C++ | mbedded.ninja

```
    tty.c_cflag &= ~CSIZE; // Clear all bits that set the data size
    tty.c_cflag |= CS8; // 8 bits per byte (most common)
    tty.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl lines (CLOCAL = 1)

    tty.c_lflag |= ICANON; // Enabling canonical mode (different from the article, we
want to read line by line)
    tty.c_lflag &= ~ECHO; // Disable echo
    tty.c_lflag &= ~ECHOE; // Disable erasure
    tty.c_lflag &= ~ECHONL; // Disable new-line echo
    tty.c_lflag &= ~ISIG; // Disable interpretation of INTR, QUIT and SUSP
    tty.c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow ctrl
    tty.c_iflag &= ~(IGNBRK|BRKINT|PARMRK|ISTRIP|INLCR|IGNCR|ICRNL); // Disable any
special handling of received bytes

    tty.c_oflag &= ~OPOST; // Prevent special interpretation of output bytes (e.g.
newline chars)
    tty.c_oflag &= ~ONLCR; // Prevent conversion of newline to carriage return/line feed

    tty.c_cc[VTIME] = TIME_OUT_VALUE_DS;    // Wait for up to 1s (10 deciseconds),
returning as soon as any data is received.
    tty.c_cc[VMIN] = 0;

    // Set in/out baud rate to be 9600
    cfsetispeed(&tty, B9600);
    cfsetospeed(&tty, B9600);

    // Save tty settings, also checking for error
    if (tcsetattr(serialPort, TCSANOW, &tty) != 0) {
        printf("Error %i from tcsetattr: %s\n", errno, strerror(errno));
        return 1;
    }

    return 0;
}
```

4.2 Reading from GPS module using serial connection

After configuring serial connection to interface with the module, the values of the module can be read using a `read()` function to read from the serial port file descriptor. For example:

```
numBytes = read(serialPort, &buf, sizeof(buf)); // Read one line from serialPort
```