Setting up and using an IMU with BBG

By: Daniel Carleton, Leighton Lagerwerf, Owen Coukell, Saman Waraich

This document guides the user through

1. Setting up the Gikfun GY-521 MPU6050 IMU on BBG
2. Running the IMU and getting gyroscope and accelerometer values

Table of Contents

Formatting

1. Commands for the host Linux's console are shown as:
   ```
   (host) $ echo "Hello PC world!"
   ```
2. Commands for the target (BeagleBone) Linux's console are shown as:
   ```
   (bbg) $ echo "Hello embedded world!"
   ```
3. Almost all commands are case sensitive

1. Setup the IMU

1.1. Wire the IMU

1. If you have not done so already, solder your pins to the device in the orientation of your choosing as long as the longer ends are not above the board on the component side

2. The Gikfun GY-521 MPU6050 IMU only needs four of the eight pins to output the values of the accelerometer and gyroscope
   Pin 1: VCC
   - This particular MPU 6050 model runs on 3-5 volts although it is recommended that you use 3.3V; wire that from the board to the pin on your IMU

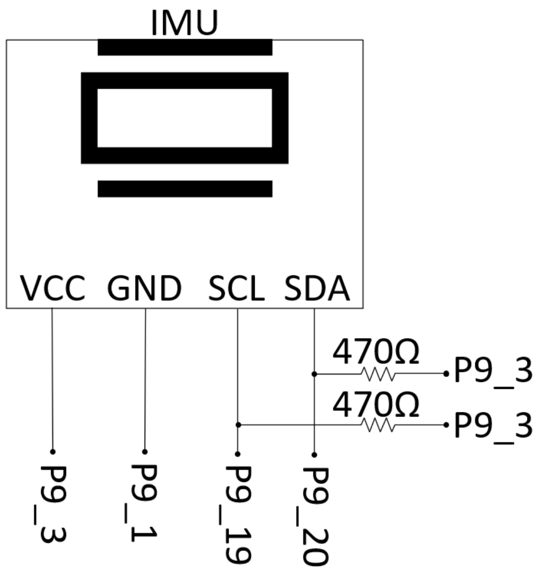
   Pin 2: GND
   - Find a ground pin on your board and connect to this pin, if using the previously recommended 3.3V on the third or fourth BeagleBone P9 pins, it is recommended that you also use either pin 1 or 2 on P9 to keep things contained

   Pin 3: SCL*
   - The Serial Clock pin on the IMU has an internal pull-up resistor, but that alone is not enough to properly maintain a signal. Still create a connection between an open I2C bus SCL pin on the BBG, just look at the below diagram for how to properly create that connection whilst implementing the pull-up resistor

   Pin 4: SDA*
   - The Serial Data pin on the IMU also has an internal pull-up resistor, but like the SCL pin, cannot properly maintain a signal without adding an additional one. Like the previous pin, find an open SDA pin connected to your I2C bus and connect using the below pull-up resistor connection

*for the diagram below as well as future steps, it is assumed you have wired to P9_19 and P9_20, using i2C bus 2 (/dev/i2c-2)

3. Schematic



1.2. Download corresponding github and default libraries

1. Once your IMU is successfully connected to the board (indicated by a green light for power), enable connection between your VM and the BBG and input commands below to initialize your pins to be in i2c mode:
   ```
   (bbg)$ config-pin P9_19 i2c
   (bbg)$ config-pin P9_20 i2c
   ```
   Those two P9 pins may be different if you used a different SCL and SDA port, change them out accordingly

2. Now go to the directory in which you wish to use the IMU on your BBG; this is where you will be installing a local library for functions and interaction with the IMU as well as basic i2c libraries that are used in the local library
```
     (bbg)$ sudo apt install -y git make gcc libc6-dev
libi2c-dev
     (bbg)$ git clone https://github.com/aler9/sensor-imu
```

1.3. Troubleshooting

1. If running the git clone command doesn't work, it's likely you do not have git installed Run:
   ```
   (bbg)$ sudo apt-get install git
   ```

2. Run the IMU

2.1. Using library functions and test code

1. Use the following testcode to make sure your IMU is reading correctly, explanations of the used commands will be shown below

main.c:

```c
#include <stdio.h>
#include <stdlib.h>

#include "sensor-imu/imu.h"

int main() {
    imut *imu;
    error *err = imu_init(&imu, "/dev/i2c-2", IMU_ACC_RANGE_2G,
IMU_GYRO_RANGE_250DPS);
    if(err != NULL) {
        printf("ERR: %s\n", err);
        return -1;
    }

    imu_output io;
    err = imu_read(imu, &io);
    if(err != NULL) {
        printf("ERR: %s\n", err);
        return -1;
    }

    printf("gyro x,y,z: %f, %f, %f\n", io.gyro.x, io.gyro.y, io.gyro.z);
    printf("acc x,y,z: %f, %f, %f\n", io.acc.x, io.acc.y, io.acc.z);
    return 0;
}
```

* Code above is sourced from the github library installation link https://github.com/aler9/sensor-imu

2. Build this with:
   ```
   (bbg)$ gcc -o main sensor-imu/*c main.c
   ```

3. Run with:
   ```
   (bbg)$ ./main
   ```

4. Expected output is something like this:

```
debian@user-beagle:/mnt/remote/myApps/IMU$ gcc -o main
sensor-imu/*.c main.c
debian@user-beagle:/mnt/remote/myApps/IMU$ ./main
found MPU6000/MPU6050 with address 0x68
gyro x,y,z: 1.403809, 0.953674, 0.228882
acc x,y,z: -0.074707, -0.019043, 0.880615
```

2.2. Using commands outside of testcode

1. Since this guide uses an online github library, the commands are exactly the same as what you would read if you looked through the github library itself
    a. You have 3 different data types that are made within the library that must be used to interact with the IMU:
        i. **imut**: used in initialization of the IMU as well as any future data collections as it stores the address of the IMU once correctly initialized
        ii. **imu_output**: used to hold the most recently received IMU data, taken as an argument in imu_read() along with the imut type
        iii. **error**: used in both IMU initialization as well as reading the IMU. Should an invalid result take place, use a printf("ERR: %s\n", [name of error var]) statement to return the error

    b. The first function you need in order to interact with the imu is "**error *imu_init(imut **pobj, const char *path, imu_acc_range acc_range, imu_gyro_range gyro_range)**" which takes in four arguments
        i. An addressof imut type variable (since imut variables should always be initialized as pointers)
        ii. The i2c bus location written as "/dev/i2c-[numberOfi2cBus]" where [numberOfi2cBus] is replaced by either 1, 2, or 3 depending on what bus you decide to use
        iii. IMU_ACC_RANGE_2G which is defined in the sensor-imu/imu.h file; refers to the range of the accelerometer
        iv. IMU_GYRO_RANGE_250DPS which is defined in the sensor-imu/imu.h file; refers to the range of the gyroscope

    c. After initializing both the imut variable and the error variable (using error *imu_init(imut **pobj, const char *path, imu_acc_range acc_range, imu_gyro_range gyro_range)), initialize your imu_output variable

d. Again using an error variable to catch any issues, you can now use **error *imu_read(imut *obj, imu_output *out)** which takes in two arguments
    i. An imut variable to point to the location of the imu
    ii. A addressof imu_output variable to store the gyro and accel data

2. You can then print all of the data from that instance of error *imu_read(imut *obj, imu_output *out) by using [your_imu_output_variable].[gyro_or_accel].[x_y_or_z] to access the values stored in your imu output variable

3. After finishing with your imu session or moving onto a piece of code that does not require it, it is highly recommended that unlike the github code provided that an **void imu_destroy(imut *obj)** to properly destroy the imu connection

4. Note: all accelerometer readings will be expressed in [g] while gyro is in [deg/s]

2.3. Troubleshooting

1. If you get the error message “ERR: no IMU sensor found” it is likely that your i2c bus is not correctly identified, make sure your pins line up with your chosen i2c bus

2. If you get the message of ERR: write() failed or ERR: read() failed, that is an error that will very rarely occur and can be ignored with a high enough sampling rate, this does not mean your IMU is unusable, but it may stutter for a small amount of time, make sure you have a decent amount of time between each sample to mitigate any requests of calling the same function twice simultaneously