TECHNICAL UNIVERSITY OF MADRID
SCHOOL OF TELECOMMUNICATIONS SYSTEMS AND ENGINEERING

Semester Project

Bc. et Bc. Jaroslav Svoboda
Andrea Vallejo
Diallo Elhadj Sadou

BeagleBone Black Project

ADVANCED DIGITAL ARCHITECTURES

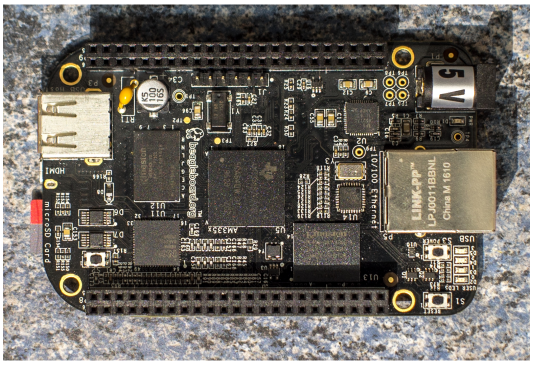
Lecturer: Eduardo Juárez
Subject coordinator: Mariano Ruiz
Degree programme: Master in Systems and Services Engineering for the Information society

Madrid 2018

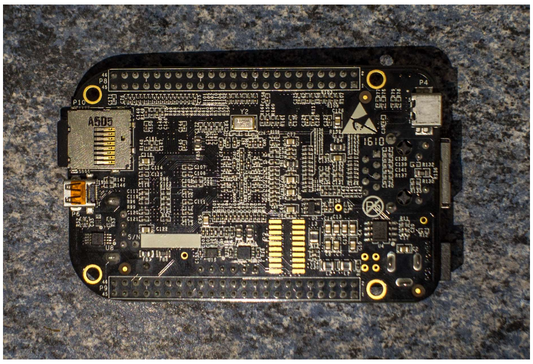
Contents

1. Embedded Linux

Whole project was done using Ubuntu GNOME 14.04.5 LTS Trusty Tahr, running in VMware Workstation 14.1.1, hosted by Kali Linux 2017.2. First we had to compile embedded Linux using Buildroot maintained by Peter Korsgaard, a tool simplifying and automating the building process of bootable Linux system for embedded solutions using cross-compilation for architectural independence. When using Buildroot we followed instructions given us in the document for lab. We set parameters accordingly to our build target BeagleBone Black which uses Texas Instruments Sitara AM3358, an ARMv7-A processor with one Cortex-A8 core. Our build uses custom Linux kernel 3.12 which includes patches for Texas Instruments SoCs. Linux is booted using U-Boot 2016.03 for processors from AM335x series. Build includes gdbserver for remote debugging, openssh for remote connection and set of elementary programs BusyBox. Two users were created: prdel and root both with password ada.



(a) BeagleBone Black top



(b) BeagleBone Black bottom

After successful compilation we created two partitions on SD card: FAT32 boot partition which includes the X-Loader (MLO), U-boot binary (u-boot.bin), Linux kernel (zImage) and device tree binaries (*.dtb), and Linux filesystem partition which is created from rootfs.ext2 file. This was one of the tricky parts and required several attempts.

When BBB successfully booted, several settings was changed using root user. Firstly, in settings of ssh deamon, file /etc/ssh/sshd_config, root user was allowed to login with password.

```
PermitRootLogin yes
```

Secondly, device was configured, file /etc/network/interface, to use static IP address for direct connection to VM in order to use remote debugging.

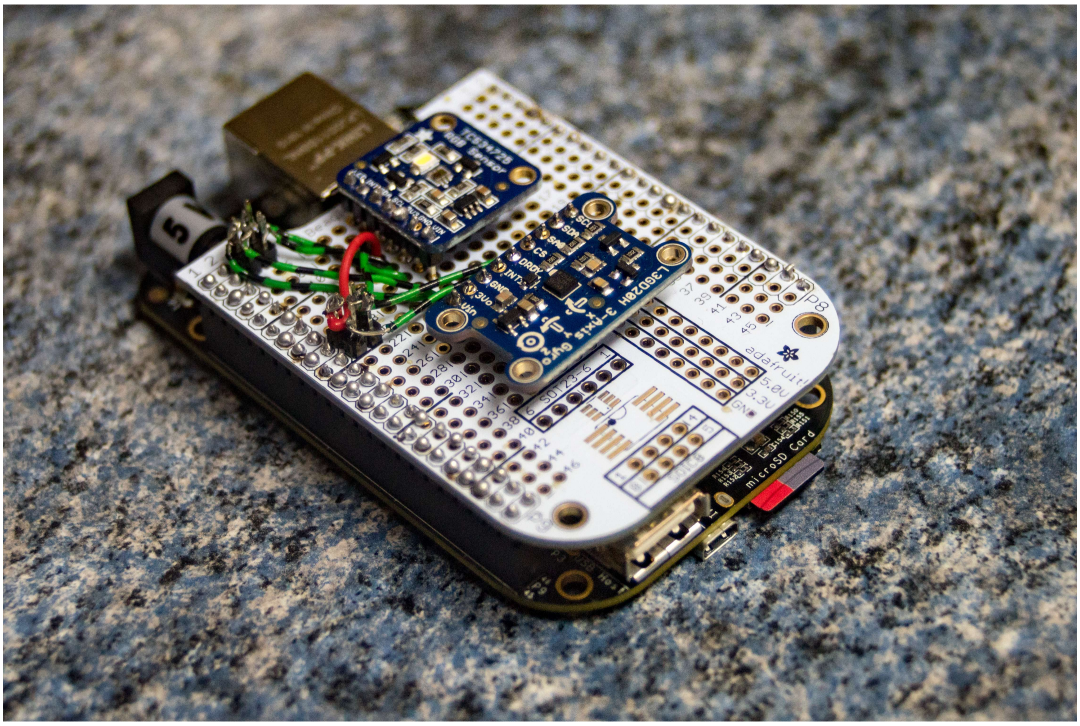
```
iface eth0 inet static
        address 192.168.1.2
        netmask 255.255.255.0
```

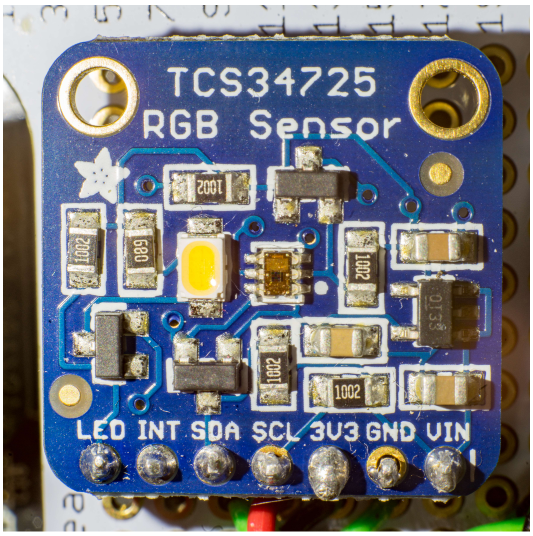
We had to rewrite the filesystem several times because it was corrupted. Other then mentioned deviations we followed the provided document for labs [2].

2. Application development and connecting of sensors

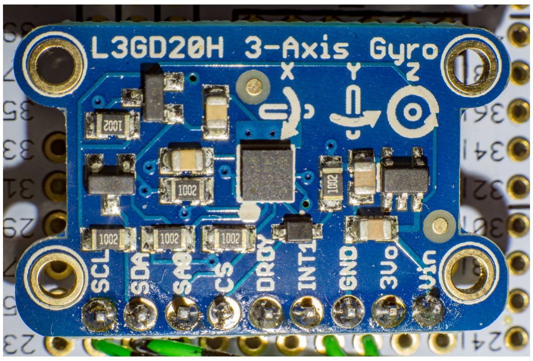
First step in connecting the TCS34725 RGB colour sensor and the L3GD20H 3-axis gyroscope was soldiering them onto HAT for practical reasons. Sensors use ground aka pins 1 and 2, 3.3 V from pins 3 and 4 for power, pins 17 and 19 for I^2C clock line and pins 18 and 20 for I^2C data line.

Figure 2.1: BeagleBone Black with HAT with sensors





(a) TCS34725 RGB colour sensor

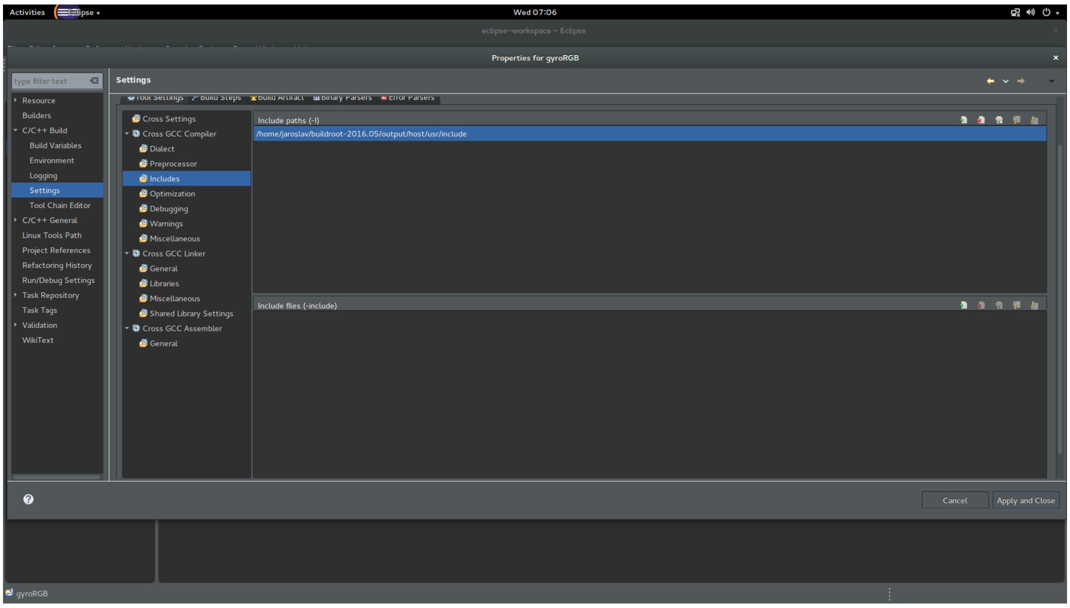


(b) L3GD20H 3-axis gyroscope

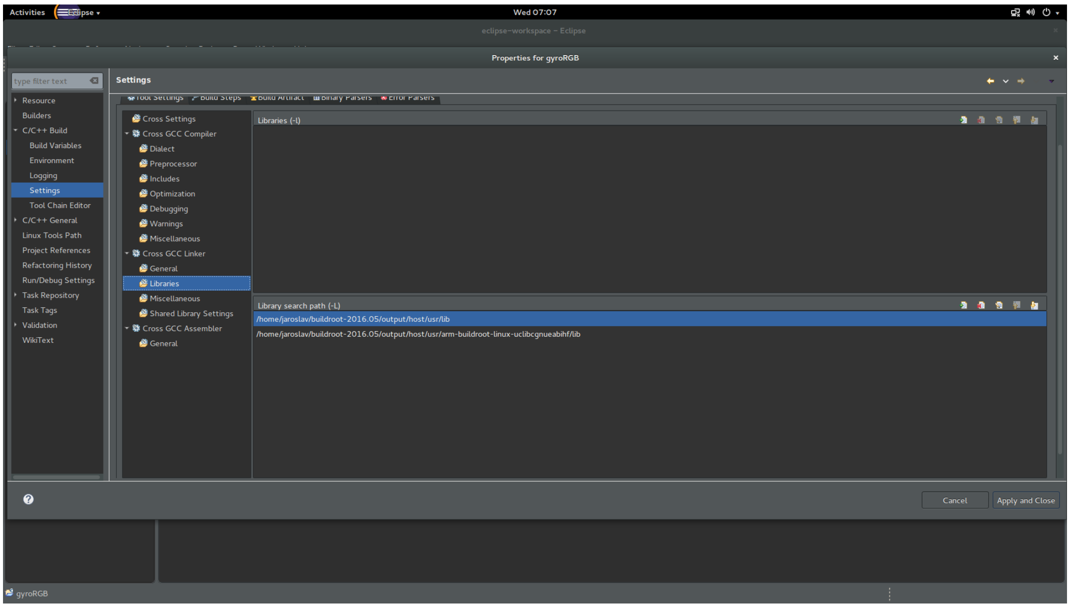
For development in C we used Eclipse Oxygen from October 2017. We set up remote debugging and cross compiling according to instructions for lab document. We used direct connection over Ethernet for remote debugging when both devices were in 192.168.1.0 subnet.

Developing and remote debugging in Eclipse Oxygen was not without complications and is definitely not stable and reliable.

First, for cross-compilation we created a workspace location. Inside this workspace, we created our Eclipse C/C++ project called: gyroRGB. After, we set the configuration of the toolchain by selecting Cross GCC. We used a Debug [Activate] configuration, as well as the GNU Make Builder. In addition, we had to put the location paths of the toolchain, the libraries and the includes. The following two figures show the whole configuration:



(a) Includes



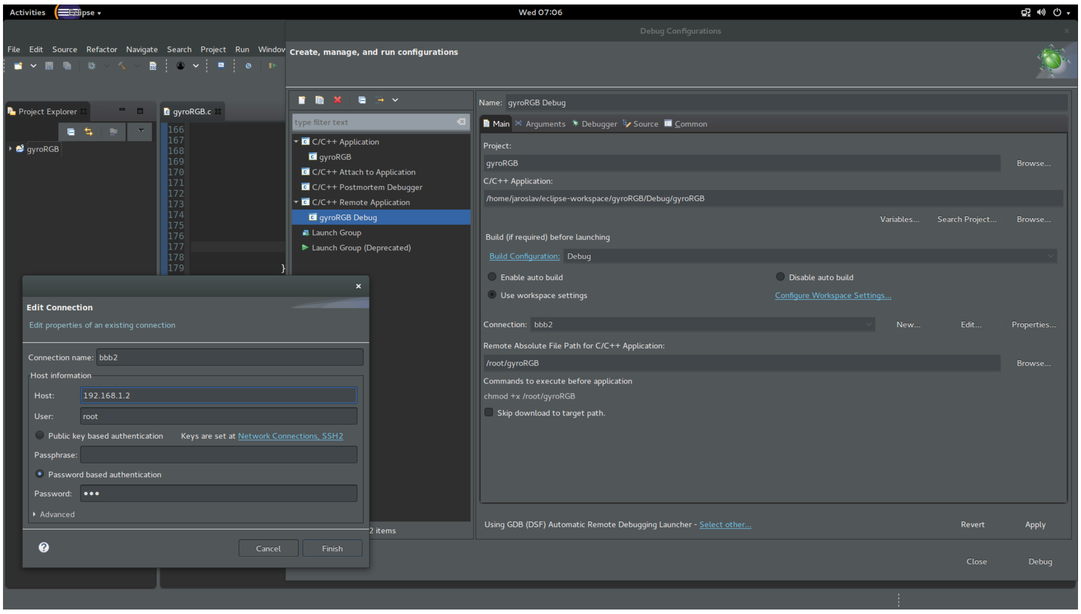
(b) Libraries

Once the code was compiled, it was necessary to debug it to be sure that we did not make any mistakes when we were coding. The configuration window is shown in the following image:

As it is possible to see there, we set the project name, the IP address of our BBB: 192.168.1.2, the remote path of the directory and application. In addition, we select an Automatic debugging mode and a root as our user profile. Moreover, we set the paths of the gdb and gdb server, which allows connecting our program with a remote GDB via target remote, our BBB. Once the configuration has been completed, we debugged our code.

At the beginning we could not finish the debugging process successfully since we got several error messages in the console window. Thanks to the breakpoint tool, we were able to find

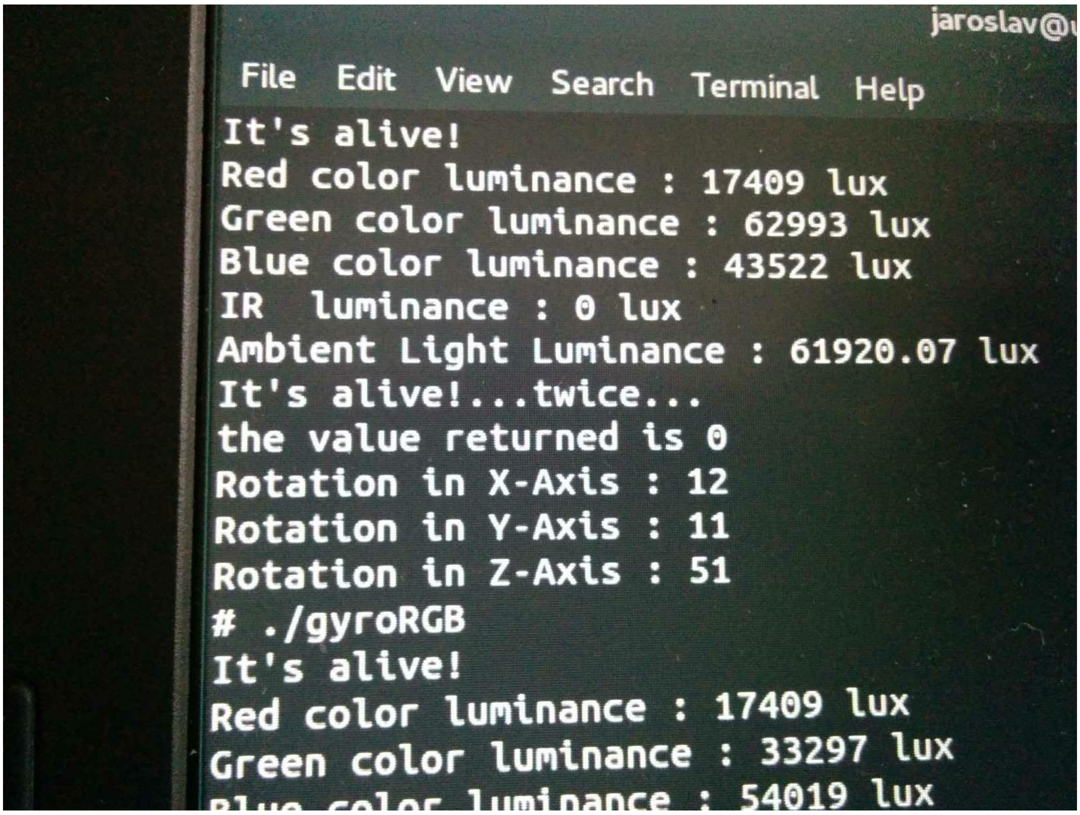
Figure 2.4: Remote debugging



that the errors came from a wrong register address allocation. Therefore we had to review the datasheet of both sensors [1, 3] in order to allocate registers available for our purpose. In addition, we found some small syntax errors and variables that have been named with the same name for different functions.

After all the mistakes were fixed, we got the following message in the terminal:

Figure 2.5: Terminal



Our RGB sensor and gyroscope were alive. We made some attempts with the RGB sensor and discovered that it worked correctly. For example, we covered the sensor with a red paper and the sensor recognized the colour red, as the image above shows. So it worked correctly. However, we found problems when we were testing the gyroscope. The values we obtained were not configured with respect to a reference. That is why we did not obtain zero for all the Cartesian coordinates X, Y, Z when the gyro was in the (0,0,0) position. We rewrite part of the code to fix this anomaly behaviour, but we could find the perfect solution, since sometimes it worked and some times do not.

In application we first initialize the I^2C bus and open it for reading to find out if the I^2C is available. The TCS34725 sensor address is $0x29$ as mentioned in [1].

```
// Create I2C bus
int file;
int file2;
char *bus = "/dev/i2c-1";
char *bus2 = "/dev/i2c-2";

if ((file = open(bus, O_RDWR)) < 0)
{
        printf("Failed to open the bus. \n");
        exit(1);
}
else
{ //the bus of RGB sensor was successfully opened
        printf("It's alive! \n");



        // Get I2C device, TCS34725 I2C address is 0x29(41)
        ioctl(file, I2C_SLAVE, 0x29);
```

According to Linux description of the command the ioctl() system call manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g. terminals) may be controlled with ioctl() requests.

We refer to the table in [1] to configure the table config we have created as you can see the address on the top and the role of each register. With this method we initialize other registers.

```
// Select enable register(0x80)
// Power ON, RGBC enable, wait time disable(0x03)
char config[2] = {0};
config[0] = 0x80;
config[1] = 0x03;
write(file, config, 2);

// Select ALS time register(0x81)
// Atime = 700 ms(0x00)
config[0] = 0x81;
config[1] = 0x00;
write(file, config, 2);
// Select Wait Time register(0x83)
// WTIME : 2.4ms(0xFF)
config[0] = 0x83;
config[1] = 0xFF;
write(file, config, 2);
// Select control register(0x8F)k
// AGAIN = 1x(0x00)
config[0] = 0x8F;
config[1] = 0x00;
write(file, config, 2);
sleep(1);
```

For the last part we just needed to know the formula of the luminance with the weight of the colour luminance for each colour.

```
// Calculate luminance
float luminance = (-0.32466) * (red) + (1.57837) * (green) + (-0.73191) * (blue)
    ;
if(luminance < 0)
{
        luminance = 0;
}

// Output data to screen
```

```
printf("Red color luminance : %d lux \n", red);
printf("Green color luminance : %d lux \n", green);
printf("Blue color luminance : %d lux \n", blue);
printf("IR  luminance : %d lux \n", cData);
printf("Ambient Light Luminance : %.2f lux \n", luminance);
}
```

Project, application, presentation and this document, is available in Git repository `https://github.com/multiflexi/adamadrina.git`

3. Code

```
// Distributed with a free-will license.
// Use it any way you want, profit or free, provided it fits in the licenses of
    its associated works.
// adapted by Andrea Vallejo, Jaroslav Svoboda and Diallo Sadou

#include <stdio.h>
#include <stdlib.h>
#include <linux/i2c-dev.h>
#include <sys/ioctl.h>
#include <fcntl.h>


int main()
{
// Create I2C bus
int file;
int file2;
char *bus = "/dev/i2c-1";
char *bus2 = "/dev/i2c-2";

if ((file = open(bus, O_RDWR)) < 0)
{
printf("Failed to open the bus. \n");
exit(1);
}
else
{ //the bus of RGB sensor was successfully opened
printf("It's alive! \n");



// Get I2C device, TCS34725 I2C address is 0x29(41)
ioctl(file, I2C_SLAVE, 0x29);

// Select enable register(0x80)
// Power ON, RGBC enable, wait time disable(0x03)
char config[2] = {0};
config[0] = 0x80;
config[1] = 0x03;
write(file, config, 2);

// Select ALS time register(0x81)
// Atime = 700 ms(0x00)
config[0] = 0x81;
config[1] = 0x00;
write(file, config, 2);
// Select Wait Time register(0x83)
// WTIME : 2.4ms(0xFF)
config[0] = 0x83;
config[1] = 0xFF;
write(file, config, 2);
// Select control register(0x8F)k
// AGAIN = 1x(0x00)
config[0] = 0x8F;
config[1] = 0x00;
write(file, config, 2);
sleep(1);

// Read 8 bytes of data from register(0x94)
// cData lsb, cData msb, red lsb, red msb, green lsb, green msb, blue lsb, blue
    msb
char reg[1] = {};
write(file, reg, 1);
char data[8] = {0};
if(read(file, data, 8) != 8)
{
printf("Erorr : Input/output Erorr \n");
}
else
```

```
{
// Convert the data
int cData = (data[1] * 256 + data[0]);
int red = (data[3] * 256 + data[2]);
int green = (data[5] * 256 + data[4]);
int blue = (data[7] * 256 + data[6]);

// Calculate luminance
float luminance = (-0.32466) * (red) + (1.57837) * (green) + (-0.73191) * (blue)
    ;
if(luminance < 0)
{
luminance = 0;
}

// Output data to screen
printf("Red␣color␣luminance␣:␣%d␣lux␣\n", red);
printf("Green␣color␣luminance␣:␣%d␣lux␣\n", green);
printf("Blue␣color␣luminance␣:␣%d␣lux␣\n", blue);
printf("IR␣␣luminance␣:␣%d␣lux␣\n", cData);
printf("Ambient␣Light␣Luminance␣:␣%.2f␣lux␣\n", luminance);
}


//STARTS GYRO
if ((file2 = open(bus2, O_RDWR)) < 0)
{
printf("Failed␣to␣open␣the␣second␣bus.␣\n");
exit(1);
}
else
{ //the bus of gyro was successfully open
printf("It's␣alive!...twice...␣\n");

printf("the␣value␣returned␣is␣%d␣\n",ioctl(file2, I2C_SLAVE, 0x6B));

// Enable X, Y, Z-Axis and disable Power down mode(0x0F)
char config[2] = {0};
config[0] = 0x20;
config[1] = 0x0F;
write(file2, config, 2);
// Full scale range, 2000 dps(0x30)
config[0] = 0x23;
config[1] = 0x30;
write(file2, config, 2);
sleep(1);

// Read 6 bytes of data
// lsb first
// for the x axe (xGyro) we are going to read the least significant bit  from
    register(0x28)
char reg[1] = {0x28};
write(file2, reg, 1);

char datai[1] = {0};
if(read(file2, datai, 1) != 1)
{
printf("Error␣:␣Input/Output␣Error␣\n");
exit(1);
}
// we get first byte in data 0
char data_0 = datai[0];

// The same we read the  most significant bit  data from register(0x29)
reg[0] = 0x29;
write(file2, reg, 1);
read(file2, datai, 1);
// we get the byte that we have read in data 1
char data_1 = datai[0];

// Read yGyro lsb data from register(0x2A)
reg[0] = 0x2A;
write(file2, reg, 1);
```

```
read(file2, datai, 1);
char data_2 = datai[0];

// Read yGyro msb data from register(0x2B)
reg[0] = 0x2B;
write(file2, reg, 1);
read(file2, datai, 1);
// same operation
char data_3 = datai[0];

// Read zGyro lsb data from register(0x2C)
reg[0] = 0x2C;
write(file2, reg, 1);
read(file2, datai, 1);
char data_4 = datai[0];

// Read zGyro msb data from register(0x2D)
reg[0] = 0x2D;
write(file2, reg, 1);
read(file2, datai, 1);
char data_5 = datai[0];

/*To convert our data and get a int value
each time we multiply by 256 that are going to decal the first bit position
to eight  rank to the left so we have 8 bits where we can add data 0

*/
int xGyro = (data_1 * 256 + data_0);
/* To avoid the possibility to have a positive  number that have more than 16
    bits
we retire 65356 to have the oposite of the number that can be in 16 bits.
so with that we have the value of the rotation for each sense of rotation
*/
if(xGyro > 32767)
{
xGyro -= 65536;
}

int yGyro = (data_3 * 256 + data_2);
if(yGyro > 32767)
{
yGyro -= 65536;
}

int zGyro = (data_5 * 256 + data_4);
if(zGyro > 32767)
{
zGyro -= 65536;
}
// Output data to screen
printf("Rotation in X-Axis : %d \n", xGyro);
printf("Rotation in Y-Axis : %d \n", yGyro);
printf("Rotation in Z-Axis : %d \n", zGyro);
}


exit(0);

}
}
```

Bibliography

[1] The LUMENOLOGY Company. *TCS3472 COLOR LIGHT-TO-DIGITAL CONVERTER with IR FILTER*. English. Texas Advanced Optoelectronic Solutions Inc. Aug. 2012. URL: `https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf` (visited on 15/01/2018).

[2] Mariano Ruiz and Francisco Javier Jiménez. *Embedded Linux Systems. Using Buildroot for building Embedded Linux Systems (BeagleBone Black)*. English. Version 1.2. ETSIS.Telecomunicación – Universidad Politécnica de Madrid. 2016. URL: `https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/1182878/mod_resource/content/1/BBBembeddedLinuxSystems6.pdf` (visited on 17/01/2018).

[3] STMicroelectronics. *MEMS motion sensor: three-axis digital output gyroscope. Datasheet - production data*. English. Version 2. 2013. URL: `https://www.pololu.com/file/0J731/L3GD20H.pdf` (visited on 17/01/2018).