PROJECT REPORT ON

TILT SENSOR APPLICATION USING MPU6050 INTERFACED WITH BEAGLEBONE BLACK

SUBMITTED IN PARTIAL
FULFILLMENT OF
THE REQUIREMENTS OF
THE AWARDS OF THE

PG-DIPLOMA IN EMBEDDED SYSTEM DESIGN

OFFERED BY
CDAC-HYDERABAD

BY

BHEEMSEN BAHARTI	**230950330005**
BURAGADDA LOKESH	**230950330006**
BURRA VINAY KUMAR	**230950330007**
C APARNA ABHISHREE	**230950330008**

ADVANCED COMPUTING TRAINING SCHOOL
C-DAC
HYDERABAD-500005

September 2023

CERTIFICATE

This is to certify that this is a Bonafide record of project entitled **"Tilt Sensor Application using MPU6050 interfaced with BeagleBone Black"**. BHEEMSEN BAHARTI-230950330005, BURAGADDA LOKESH -230950330006, BURRA VINAY KUMAR-230950330007 and C APARNA ABHISHREE-230950330008, have completed project work as part of Diploma in Embedded System Design (SEPTEMBER,2023-Batch), a PG course offered by C-DAC Hyderabad. They have completed project work under the supervision of Mr. Prasad Nissi. Their Performance was found to be good.

ACKNOWLEGEMENT

The implementation of a Tilt Sensor application utilizing the MPU 6050 interfaced with Beaglebone Black is presented in this project. This endeavor represents a significant milestone in our journey, as we navigate through various challenges, both past, present, and those yet to come. It's imperative to acknowledge the invaluable support and guidance received from several individuals. Foremost, our sincere gratitude goes to **Mr. Prasad Nissi**, our mentor, whose unwavering assistance and counsel were instrumental in the successful execution of this project. Additionally, his efforts in sourcing all the necessary components played a pivotal role in its realization.

(PG-DESD SEPTEMBER 2023)

BHEEMSEN BAHARTI	**230950330005**
BURAGADDA LOKESH	**230950330006**
BURRA VINAY KUMAR	**230950330007**
C APARNA ABHISHREE	**230950330008**

TABLE OF CONTENTS

ABSTRACT

Tilt sensing is an indispensable aspect across various industries and applications, impacting stability, safety, accuracy, control, efficiency, and user experience. It plays a pivotal role in automotive applications by ensuring vehicle stability, preventing rollovers, and enhancing handling performance. In photography and satellite communication, tilt sensors enable precise alignment, leading to superior image quality and reliable communication links. Similarly, in UAVs, tilt sensing stabilizes flight, controls navigation, and facilitates precise aerial maneuvers. Industrial and agricultural machinery benefit from tilt sensors by optimizing productivity, ensuring stability on uneven terrain, and minimizing accidents. Consumer electronics incorporate tilt sensing for features like screen rotation and motion-controlled interactions, enhancing user convenience and satisfaction. Overall, tilt sensing technology enhances operational performance and safety across a wide spectrum of applications and industries.

The primary objective of the project is to determine the tilt angle by utilizing the MPU6050 sensor in conjunction with the BeagleBone Black Microcontroller. The MPU6050 sensor employs the I2C protocol to establish communication with the BeagleBone Black. Its 3-axis Accelerometer transmits the initial X, Y, and Z acceleration data, which are subsequently translated into 'g' values by selecting the appropriate full-scale sensitivity. Similarly, the 3-axis gyroscope provides the initial tilt values across the X, Y, and Z axes, which are then converted into degrees per second for pitch, roll, and yaw, respectively. Utilizing these data points, calibration of the application can be executed to ensure a balanced level of accuracy and

precision.

1. INTRODUCTION AND REQUIREMENTS

1.1Tilt sensor applications.

- To detect the correct angle at which the ground station's antenna can establish successful communication with the satellite in space.
- To control the pitch and roll angles in drones and Airplanes.
- To maintain proper orientation on uneven terrains and slopes for agriculture machinery.
- To build fall detection for elderly or workers in Hazardous working conditions.
- To monitor the angle at which a mobile phone or tablet is held for the auto-rotate function.

1.2 Software and Hardware Requirements

1.2.1 Hardware Requirements

- Development Board-BeagleBone Black Rev C
- MPU-6050 sensor
- USB to TTL converter
- SD Card

1.2.2 Software Specifications

- Operating System: Debian
- Toolchain: arm-linux-gnueabihf-
- Partition: GParted

1.3 Coding and Testing

- C language for programming
- Shell commands

2. IMPLEMENTATION

2.1BeagleBone Black Rev C

The BeagleBone Black Rev C is a specific revision of the BeagleBone Black single-board computer. Each revision, marked by a letter (e.g., Rev A, Rev B, Rev C), typically includes improvements, bug fixes, or changes to the design or components.

Example:

In rev B: Changed the processor to AM3358BZCZ100.

In rev C: Changed the eMMC from 2GB to 4GB.

2GB devices are getting harder to get as they are being phased out. This required us to move to 4GB. We now have two sources for the device. This will, however, require an increase in the price of the board.

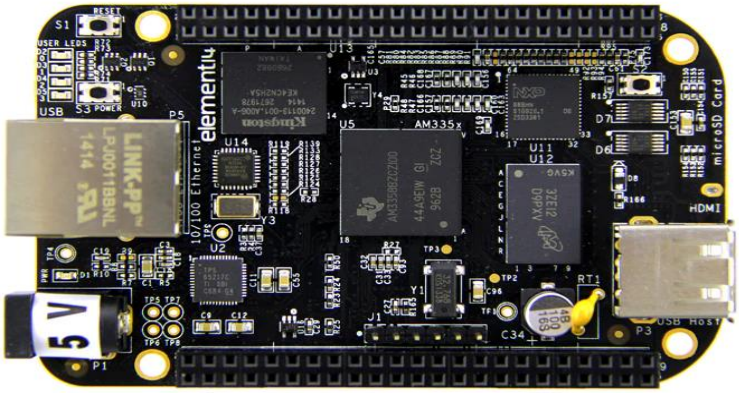

Figure 1 BeagleBone Black rev C Board

The BeagleBone Black Microcontroller specifications are as follows:

➤The processor type – Sitara AM3358BZCZ100 with 1 GHz and 2000 MIPS

➤Graphics Engine- 20M Polygons/S, SGX530 3D

➤Size of SDRAM memory – 512 MB DDR3L, 800 MHz

➤Onboard Flash- 8-bit Embedded MMC with 4 GB

➤PMIC – 1 additional LDO, TPS65217C PMIC regulator

➤Debug Support – Serial Header, onboard optional 20-pin CTI

➤Power Source – mini-USB, USB or DC jack; 5 Volts external DC through expansion header

➤PCB – 3.4″ X 2.1″; 6 layers

➤Type of indicators – 1 power, 2 Ethernet, 4 LEDs, which are user-controllable

➤HS USB 2.0 Host Port – Accessible to USB1, Type A Socket, 500 mA LS/FS/HS

➤Serial Port – UART0 access via 6-pin 3.3 Volts TTL header. Populated header

➤Ethernet – 10/100, RJ45

➤User Input – Power button, reset button, Boot button

➤SD/MMC Connector – microSD, 3.3 Volts

➤Video out – 16b HDMI, 1280×1024 (max), 1025×768, 1280×720, 1440×900, w/EDID support

➤HS USB 2.0 Client Port – Access to USB0, client mode through mini-USB

➤Audio – Stereo, via HDMI interface

➤Weight – 39.68 gms (1.4 oz)

➤Expansion Connectors – 5 Volts, 3.3 Volts power, VDD_ADC 1.8 Volts.

3.3 Volts on all I/O signals – GPIO (69 max), McASP0, I2C, SPI1, LCD, GPMC, MMC1, MMC2, 4 serial ports, 4 timers, 7 AIN (max 1.8 Volts), CAN0, XDMA

interrupt, EHRPWM (0, 2), Power button, expansion board ID (stacking of up to 4). The board is designed to use the Sitara AM3358BZCZ100 processor in the 15 x 15 package. Earlier revisions of the board used the XM3359AZCZ100 processor.

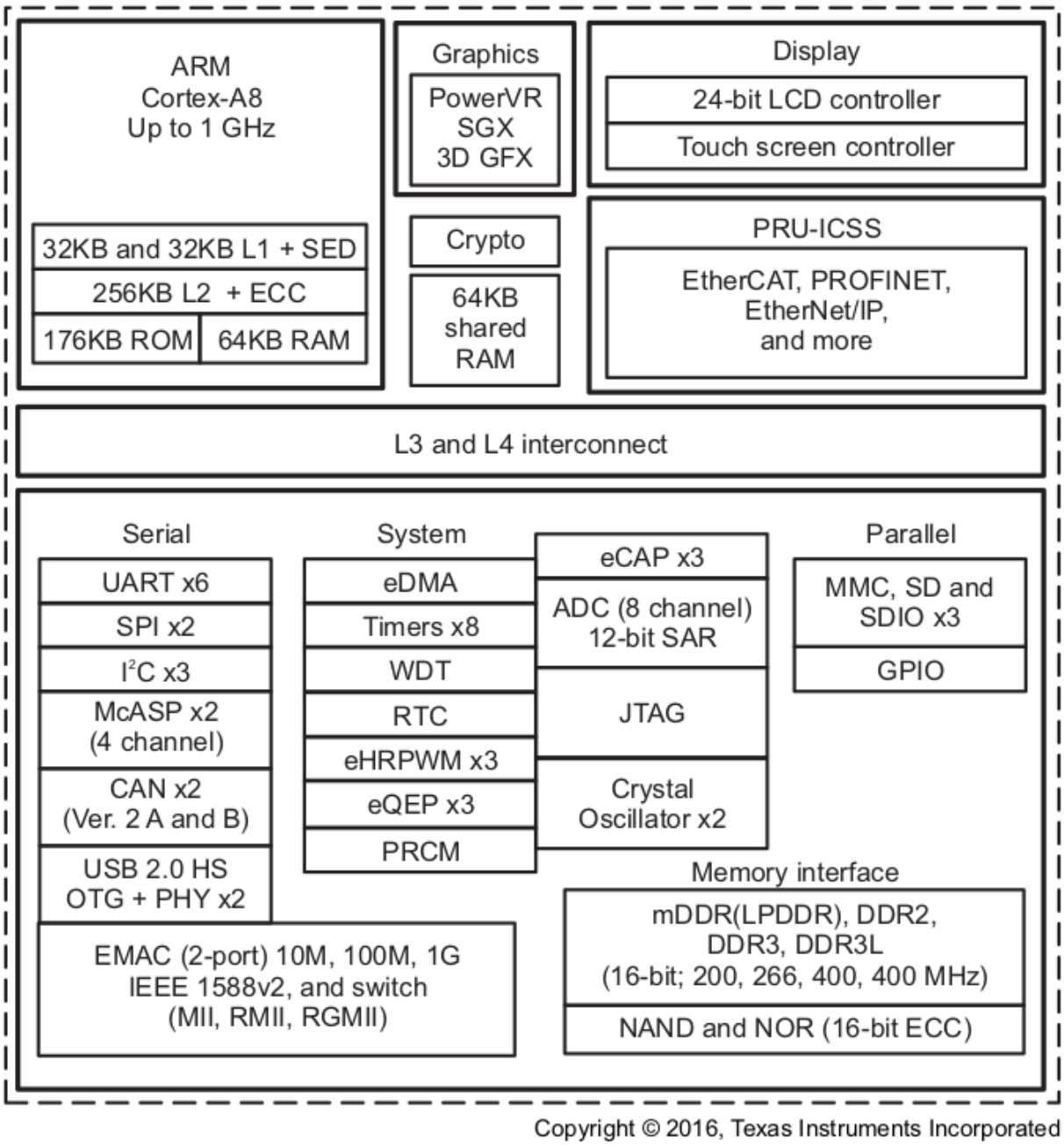

Figure 2 Sitara AM3358BZCZ Block Diagram

2.2 USB to TTL Converter

BeagleBone Black board consists of a serial debug port which is provided via UART0 on the processor via a single 1x6 pin header. The Serial Debug Port on the BeagleBone Black is typically used for debugging purposes and for accessing the device's console interface. It provides a way to interact with the BeagleBone's operating system, boot loader, and other low-level functions through a serial connection.

The Serial Debug Port on the BeagleBone Black typically uses UART communication, which involves transmitting and receiving data through two pins: TX (transmit) and RX (receive). These pins operate at TTL logic levels, typically around 3.3 volts. When working with devices like the BeagleBone Black, which have a Serial Debug Port for accessing the console interface, Minicom can be used to establish a serial connection to the device. This allows users to access the device's console, view boot messages, enter commands, troubleshoot issues, and perform other debugging tasks.

Figure 3 UART to TTL Switch

So, to connect Beaglebone with the converter we require only 3 pins of the UART. The converter's ground pin (GND) links to pin 1 of the UART (J1) connector, while the converter's TXD (transmitter) pin connects to pin 4 of the same connector. Likewise, the converter's RXD (receiver) pin is linked to pin 5 of the UART (J1) connector. This setup establishes a serial communication link between the BeagleBone and our Linux system, to which the converter's USB will be attached.

Once you've linked the ground, receiver, and transmitter pins of the TTL converter to the UART – J1 pins of the BeagleBone Board, plug the USB end into the Linux host system.

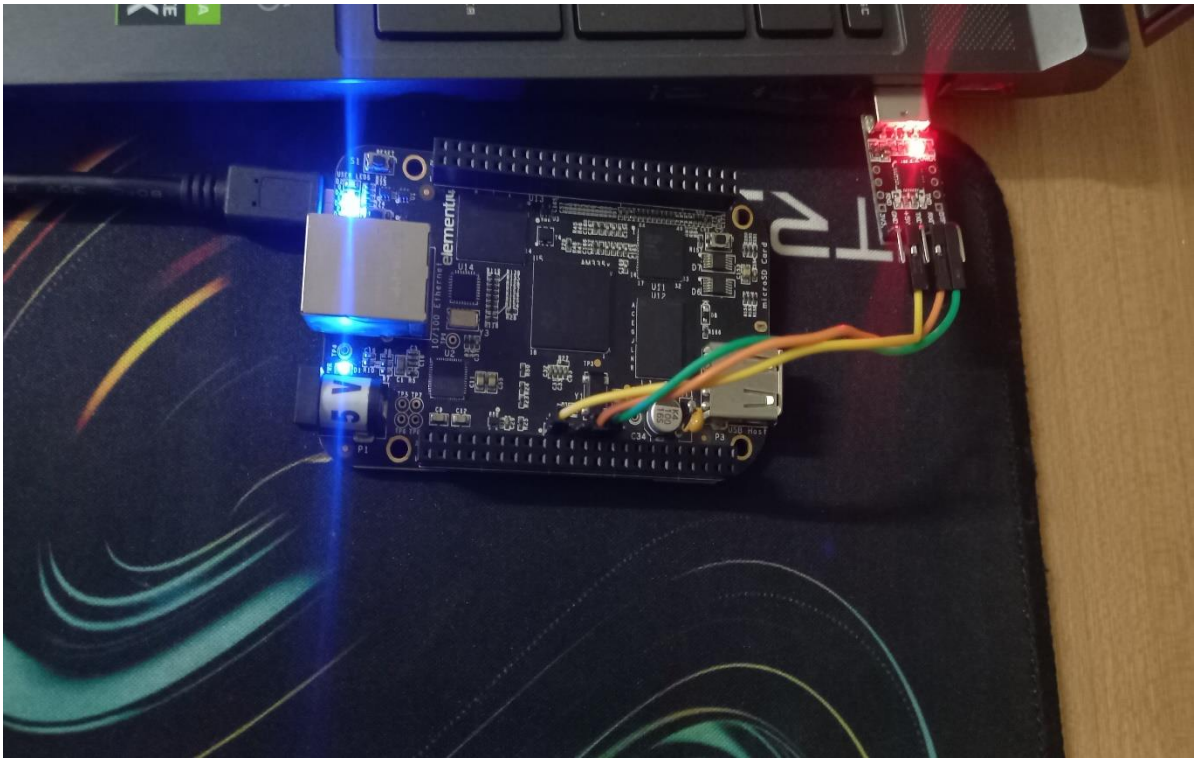

Figure 4 Connecting UART to TTL Switch to BeagleBone Black

Then, in the terminal of our Linux system, we verify the system's detection of the converter by executing the following command.

ls /dev => it lists all the devices attached to it

If you find “ttyUSB0”, then the UART to TTL is successfully connected to/detected by the host system (PC). To install minicom give the commands as:

sudo apt install minicom => installs minicom on the system

sudo minicom –s => opens the settings of minicom (s- settings to the system)

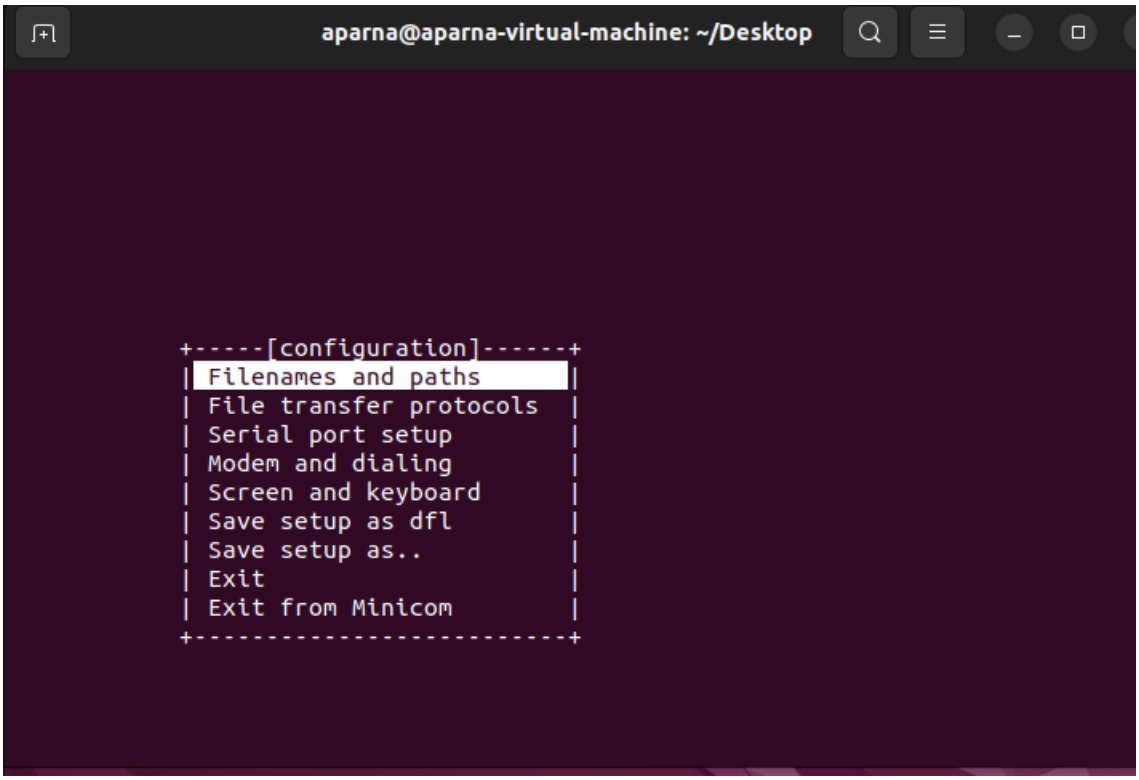

Figure 5 Minicom Set up

After giving the commands, a configuration file opens. In that go to “Serial port setup” and click enter. Set the serial device by entering “A” on keyboard. Next, we enter device as “/dev/ttyUSB0” and then we hit ENTER. Also, check whether the baud rate is 11520 to establish connection with the board and select “save as default” before hitting “exit”, which then direct to a minicom terminal window. In the minicom window, essential details about our port are now visible.

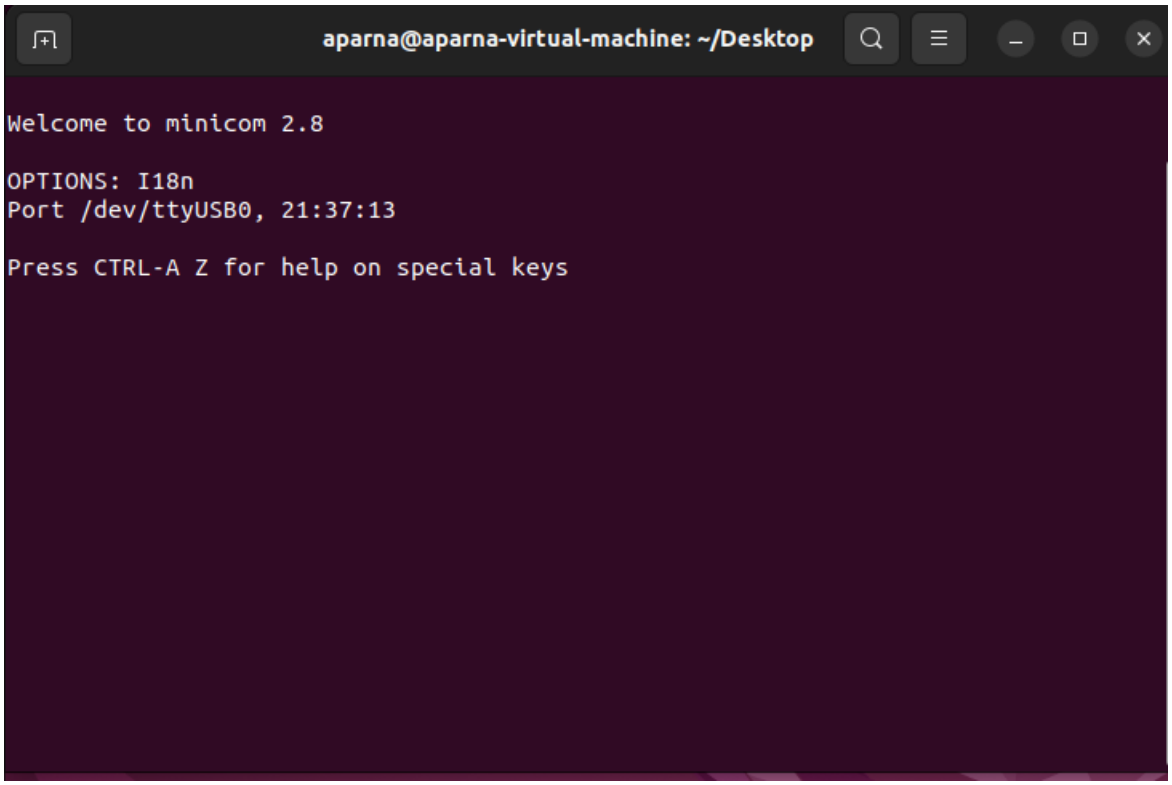

Figure 6 Minicom Window

Next, we power up the BeagleBone board. BeagleBone offers two boot options, with the default being the Debian OS loaded from the eMMC. As the bootloader and kernel start, their boot-up information appears in the minicom terminal.Towards the end of the boot process, the last 5-6 lines display key information about the board, including its IP address and the default username-password combination required to access the board's terminal. Following the boot-up sequence, we enter "debian" (the default username) and "temppwd" (the default password, as indicated on the screen). This completes the boot process, granting us access to the board's terminal.

Figure 7 U-Boot process shown in minicom window

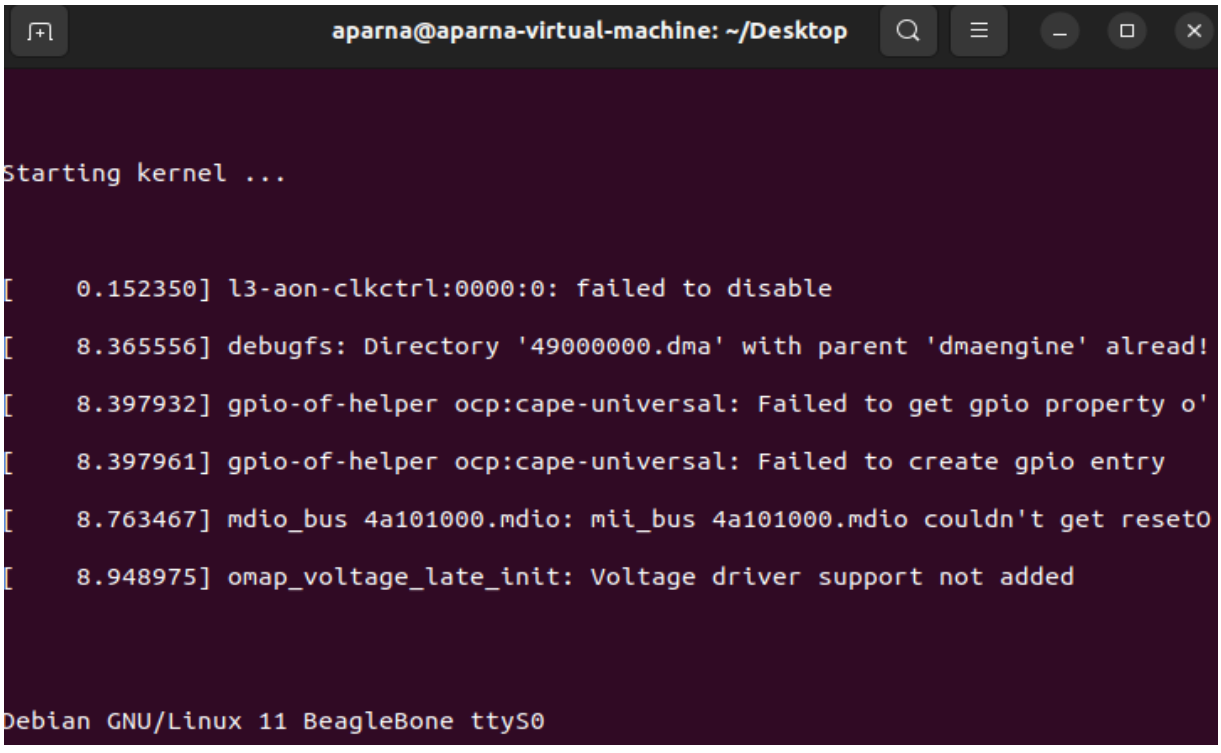

Figure 8 Loading of Kernel by U-Boot

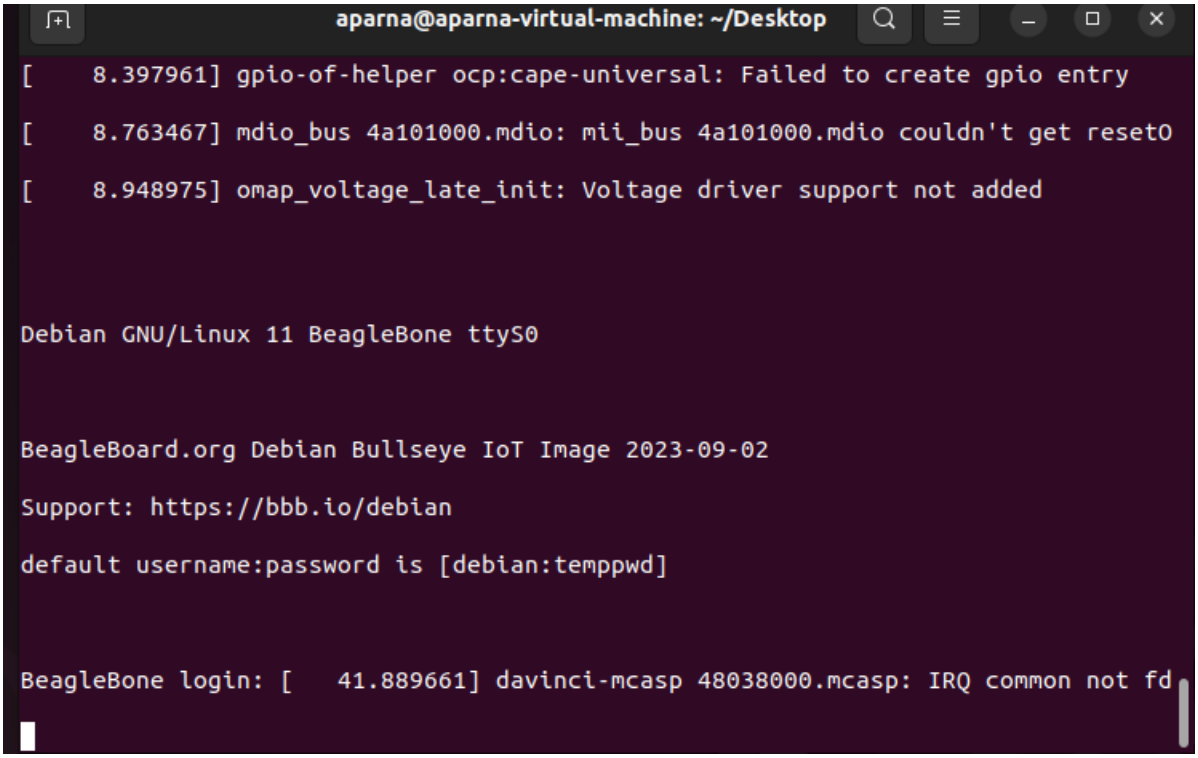

Figure 9 Board details is displayed.

2.3 Booting Sequence of BeagleBone Black

Booting Sequence without an SD Card:

Upon powering on the BeagleBone without an SD card inserted, the bootloader stored in the internal eMMC (embedded MultiMediaCard) storage takes charge. It initializes the hardware, loads the Linux kernel from the eMMC into memory, and potentially loads a device tree binary if needed. Following the kernel loading process, the init process initializes the system environment, mounts the root filesystem, and starts essential system services, ultimately transitioning to user space for interaction. A single 4GB embedded MMC (eMMC) device is on the board. The device connects to the MMC1 port of the processor, allowing for 8bit wide access. Default boot mode for the board will be MMC1 with an option to change it to MMC0, the SD card slot, for booting from the SD card because of removing and

reapplying the power to the board.

Booting Sequence with an SD Card:

When the BeagleBone is powered on with an SD card inserted, the boot process commences with the initialization of hardware components. The bootloader, stored on the SD card, is the first piece of software executed. It initializes the system, loads the Linux kernel from the specified location on the SD card into memory, and may also load a device tree binary describing the hardware configuration. Subsequently, the kernel starts the init process, which sets up the system environment, mounts the root filesystem, and launches essential system services. Finally, the system transitions to user space, allowing users to interact with the BeagleBone through a terminal or graphical interface.

The board is equipped with a single microSD connector to act as the secondary boot source for the board and, if selected as such (using Boot button), can be the primary boot source. The connector will support larger capacity microSD cards. The microSD card is not provided with the board. Booting from MMC0 will be used to flash the eMMC in the production environment or can be used by the user to update the SW as needed. For booting from SD card, first press the S2 (boot) button and then apply power to the board so that the booting can happen from the SD card.

Whichever way it is booted in, the basic power supply and led functionality is same. The basic LED sequence is same and works as follows:

- USER0 is the heartbeat indicator from the Linux kernel.
- USER1 turns on when the SD card is being accessed.
- USER2 is an activity indicator. It turns on when the kernel is not in the idle loop
- USER3 turns on when the onboard eMMC is being accessed.

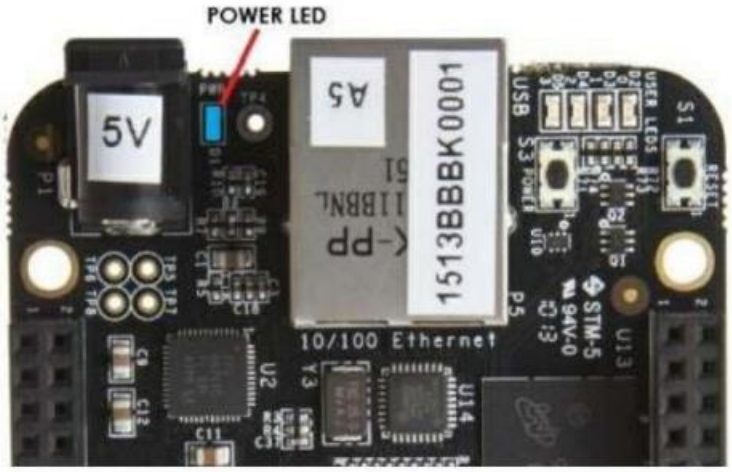

Figure 10 Power LED ON when Beaglebone is connected to power supply.

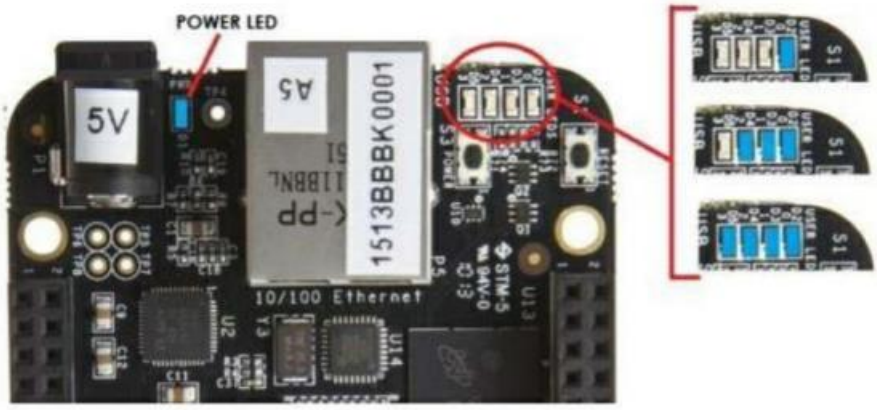

Figure 111 User LEDS glowing on the board.

2.4 MPU6050

The MPU6050 is a popular sensor module commonly used in electronics projects, especially in the fields of robotics, motion tracking, and gesture recognition. It combines a 3-axis gyroscope and a 3-axis accelerometer into a single integrated circuit, allowing it to measure both rotational and linear motion. As a beginner-friendly sensor, the MPU6050 provides precise data on orientation (roll, pitch, and yaw) and acceleration in all three dimensions. Its small size and low power

consumption make it suitable for integration into various devices. The MPU6050 communicates with microcontrollers or other devices via the I2C (Inter-Integrated Circuit) protocol, enabling easy interfacing with popular development boards like Arduino and Raspberry Pi. By processing data from the MPU6050, developers can implement features such as stabilization, gesture recognition, and motion tracking in their projects. With its versatility and ease of use, the MPU6050 is an excellent choice for beginners looking to explore the world of motion sensing and control in their electronic creations.

Figure 121 MPU6050 sensor.

The MPU6050 has registers from 0x0D to 0x75. Each of these registers are 8- bits wide and have specific functions. The accelerometer and gyroscope data are 16 bits wide and so data from each axis uses two registers high and Low.

Register	**Address**
ACCEL_XOUT_H	0x3B
ACCEL_XOUT_L	0x3C
ACCEL_YOUT_H	0x3D
ACCEL_YOUT_L	0x3E
ACCEL_ZOUT_H	0x3F

ACCEL_ZOUT_L	0x40

Table 1 Accelerometer Register Addresses for MPU6050

Register	**Address**
GYRO_XOUT_H	0x43
GYRO_XOUT_L	0x44
GYRO_YOUT_H	0x45
GYRO_YOUT_L	0x46
GYRO_ZOUT_H	0x47
GYRO_ZOUT_L	0x48

Table 2 Gyroscope Register Addresses for MPU6050

These registers store the most recent accelerometer measurements. Each 16-bit accelerometer measurement has a full scale defined in ACCEL_FS (Register 28). First, the registers on the table don't have any numbers in them. We must wake up the MPU6050 device before it can start giving us data. To wake it up, we just need to set a certain register to zero, and that register is called PWR_MGMT_1, found at address 0x6B.

The data we get from the sensors is basic and needs to be kept within a certain range. We can control this range by adjusting two values called FS_SEL and AFS_SEL. These values determine the maximum amount we can measure. In our case, we are using the switch statement which enables the user to select sensitivity and range base on his/her choice. When we set these values, another important number pops up called the Sensitivity Scale Factor.

CONDITION	**SENSITIVITY**
ACC_FS_SENSITIVITY_0	16384
ACC_FS_SENSITIVITY_1	8192

ACC_FS_SENSITIVITY_2	4096
ACC_FS_SENSITIVITY_3	2048

Table 3. Full-Scale modes for MPU6050 Accelerometer

CONDITION	**SENSITIVITY**
GYR_FS_SENSITIVITY_0	131
GYR_FS_SENSITIVITY_1	65.5
GYR_FS_SENSITIVITY_2	32.8
GYR_FS_SENSITIVITY_3	16.4

Table 4. Full-Scale modes for MPU6050 Gyroscope

From the table above, when we choose FS_SEL=0, the Sensitivity Scale Factor becomes 131. This means that 1 degree per second of angular velocity is represented as 131 in the gyro sensor. So, to find the angular velocity in degrees per second, we divide the reading from the sensor by the sensitivity scale factor (which in this case is 131). Similarly, for the accelerometer, if we set AFS_SEL=0, the Sensitivity Scale Factor becomes 16384. This means that 1g (gravity) is represented as 16384 in the accelerometer.

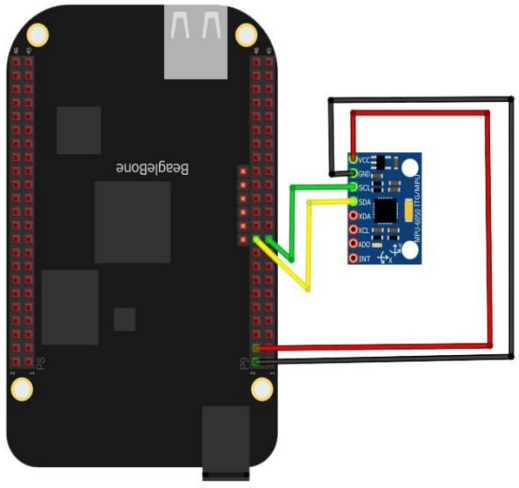

Figure 13 MPU 6050 to BBB Connection

From the above figure we can see that the MPU6050 works on I2C protocol, and the SDA SCL lines are connected to the P9_20 and P9_19 pins of the Beaglebone Black Board. The other pins are connected to the Ground and Voltage Supply of 3.3 V.

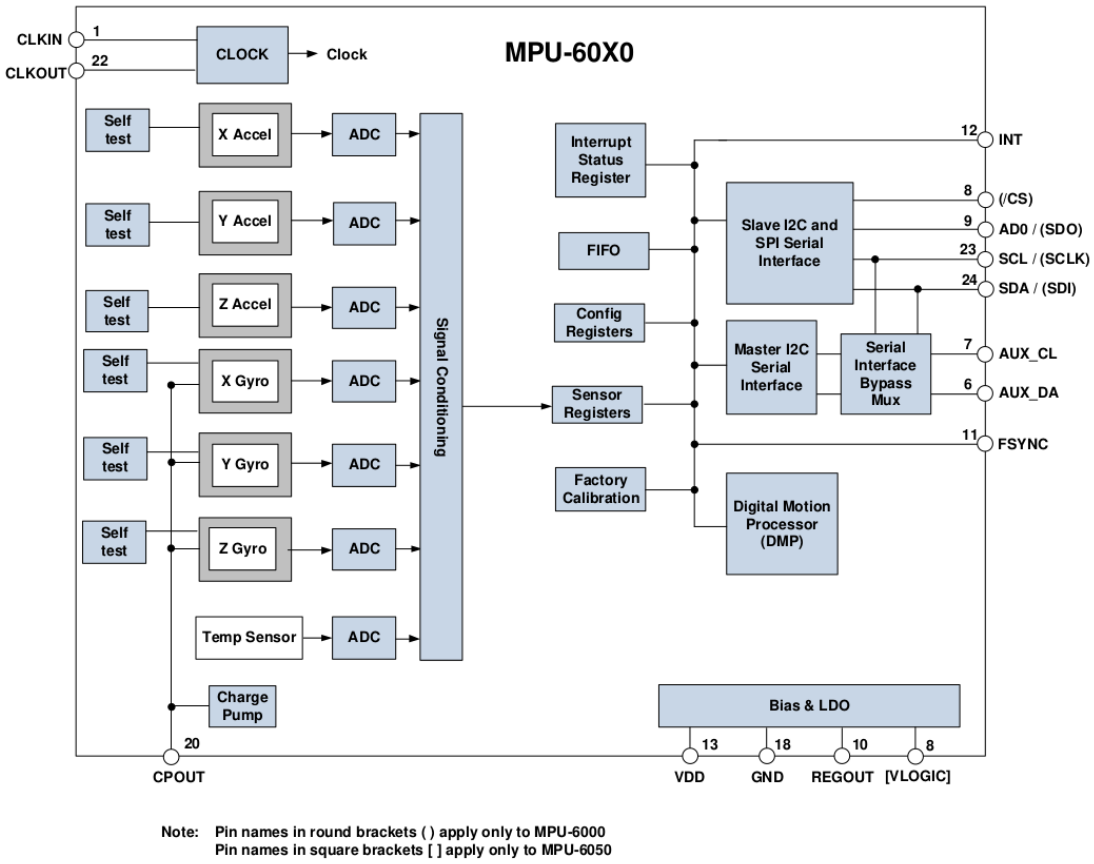

Figure 14 Internal Architecture of MPU 60X0.

2.5 I2C PROTOCOL

We refer to inter-integrated circuits as I2C. It is a bus interface connection protocol that is built into serial communication devices. It has become a popular short-distance communication protocol in recent years. Another name for it is Two Wired Interface (TWI). It exclusively communicates data via the SDA and SCL bi-directional open-drain lines. Since the devices on the I2C bus are active low, a pull-up resistor is needed to make the lines high since both lines are pulled high. Serial Data (SDA) - This pin is used for data transfer. The clock signal is carried via the

Serial Clock (SCL).

I2C functions in two modes:

- Master mode.
- Slave mode.

Working of I2C Communication Protocol:

A high to low pulse from each clock on the SCL wire synchronizes each data bit sent on SDA line. I2C protocols state that the data line can only change when the clock line is low and cannot change while the clock line is high. The information is sent in the form of nine-bit packets. By maintaining the SCL line high and varying the SDA level, START and STOP bits may be produced. The SDA is altered from high to low while maintaining a high SCL to produce the START condition. As seen in the image below, in order to establish the STOP condition, SDA increases from low to high while maintaining a high SCL. Data is sent over I2C in messages. Data frames are used to segment messages. Every message consists of one or more data frames with the data being transferred and an address frame with the slave's binary address. In addition, the message contains read/write, ACK/NACK, and start/stop conditions in between each data frame.

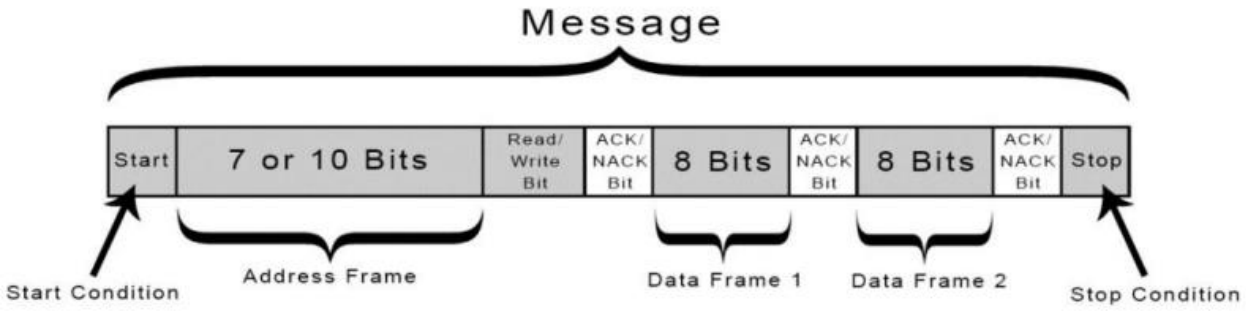

Figure 15 I2C protocol Message Frame Format

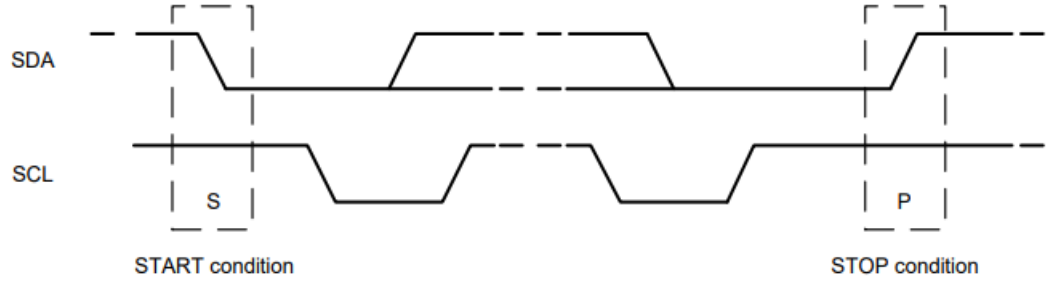

Figure 16 Start and Stop Condition

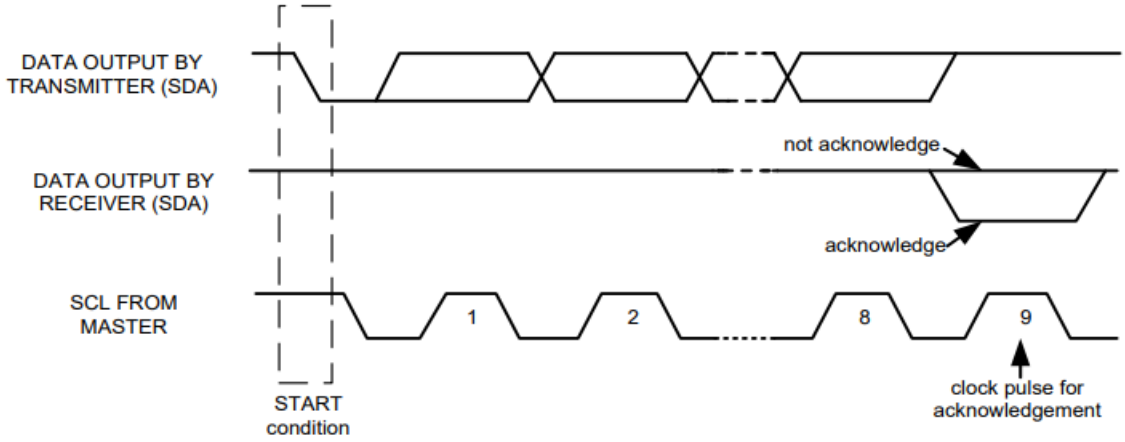

Figure 17 Acknowledgement from Slave

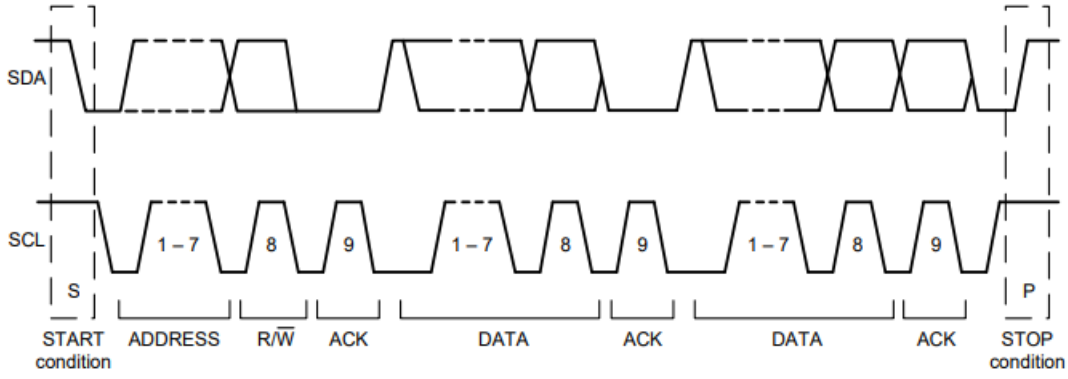

Figure 18 Data Transfer on SDA

Single-Byte Write Sequence

Master	S	AD+W		RA		DATA		P
Slave			ACK		ACK		ACK	

Burst Write Sequence

Master	S	AD+W		RA		DATA		DATA		P
Slave			ACK		ACK		ACK		ACK	

Figure 19 Frame Format for Single-Byte and Burst Write sequence.

Single-Byte Read Sequence

Master	S	AD+W		RA		S	AD+R			NACK	P
Slave			ACK		ACK			ACK	DATA		

Burst Read Sequence

Master	S	AD+W		RA		S	AD+R			ACK		NACK	P
Slave			ACK		ACK			ACK	DATA		DATA		

Figure 20 Frame Format for Single-Byte and Burst Read sequence.

Signal	Description
S	Start Condition: SDA goes from high to low while SCL is high
AD	Slave I^2C address
W	Write bit (0)
R	Read bit (1)
ACK	Acknowledge: SDA line is low while the SCL line is high at the 9^{th} clock cycle
NACK	Not-Acknowledge: SDA line stays high at the 9^{th} clock cycle
RA	MPU-60X0 internal register address
DATA	Transmit or received data
P	Stop condition: SDA going from low to high while SCL is high

Table 5 I2C Terms to remember.

There are three I2C buses on the Beaglebone Black according to the AM335X Technical Reference Manual and their memory addresses are:

i2c-0: 0x44E0_B000

i2c-1: 0x4802_A000

i2c-2: 0x4819_C000

The i2c-0 bus is not accessible on the header pins while the i2c-1 bus is utilized for reading EEPROMS on cape add-on boards and may interfere with that function when used for other digital I/O operations. Hence, we use the i2c-2 bus on pins 19 and 20.

2.6 CODE IMPLEMENTATION

First, we define all the headers required for writing the program.

```
#include<stdio.h>
#include<stdint.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<sys/ioctl.h>
#include<linux/i2c-dev.h>
```

Then, we define all the addresses required to fetch and start the MPU-6050 sensor.

```
#define MPU6050_ADDR 0x68 // MPU6050 I2C address
#define MPU6050_PWR_MGMT 0x6B //MPU6050 power management register
```

```
//mpu6050 address to get accelerometer values
#define ACCEL_XOUT_H        0x3B
#define ACCEL_XOUT_L        0x3C
#define ACCEL_YOUT_H        0x3D
#define ACCEL_YOUT_L        0x3E
#define ACCEL_ZOUT_H        0x3F
#define ACCEL_ZOUT_L        0x40

//mpu6050 address to get gyro values
#define GYRO_XOUT_H         0x43
#define GYRO_XOUT_L         0x44
#define GYRO_YOUT_H         0x45
#define GYRO_YOUT_L         0x46
#define GYRO_ZOUT_H         0x47
#define GYRO_ZOUT_L         0x48
```

We can make use of the Configuration registers to change the sensitivity scale/range to convert the raw data values into g values.

```
//address of gyroscope configuration register
#define GYRO_CONFIG         0x1B
//address of accelerometer configuration register
#define ACCEL_CONFIG        0x1C
```

```
// Different full-scale ranges for acc and gyro
#define ACC_FS_SENSITIVITY_0        16384
#define ACC_FS_SENSITIVITY_1         8192
#define ACC_FS_SENSITIVITY_2         4096
#define ACC_FS_SENSITIVITY_3         2048

#define GYR_FS_SENSITIVITY_0          131
#define GYR_FS_SENSITIVITY_1          65.5
#define GYR_FS_SENSITIVITY_2          32.8
#define GYR_FS_SENSITIVITY_3          16.4

//global declarations of file_descriptor and switch case variable
int fd;
int choice;
```

Here, we define a function to write the address and data into the MPU6050.

```
//to configure the registers of MPU6050
int MPU6050_write (uint8_t add, uint8_t data){
    char buf[2];
    buf[0]=add;
    buf[1]=data;
    if(write(fd,buf,sizeof(buf))<0){
        printf("error in the write\n");
        return -1;
    }
```

```
    return 0;
}

//user defined function to initialize the mpu6050.
```

Here we power up the MPU6050 with the help of MPU6050_PWR_MGMT register and then we try to set the accelerometer and gyroscope sensitivity by using the ACCEL_CONFIG and GYRO_CONFIG registers based on our interest of setting the sensitivity.

```
void MPU6050_init()
{
    //to wake the mpu6050 from the sleep mode
    MPU6050_write(MPU6050_PWR_MGMT,0x00);
    usleep(100000);  //sleep for 0.1 sec

    //to configure the mpu6050_accelerometer and mpu6050_gyroscope full scale sensitivity
    printf("enter your choice:\n");
    printf("1.sensitivity 0\n2.sensitivity 1\n3.sensitivity 2\n4.sensitivity 3\n");
    scanf("%d",&choice);
    switch(choice){
        case 1:
            MPU6050_write(ACCEL_CONFIG, 0x00); // For mpu6050_accelerometer sensitivity 0 (±2 g)
            usleep(100000);
```

```
        MPU6050_write(GYRO_CONFIG, 0x00); // For mpu6050_gyro sensitivity
0 (±250 º/s)
        usleep(100000);
        break;

    case 2:
        MPU6050_write(ACCEL_CONFIG, 0x08); // For mpu6050_accelerometer
sensitivity 1 (±4 g)
        usleep(100000);
        MPU6050_write(GYRO_CONFIG, 0x08); // For mpu6050_gyro sensitivity
1 (±500 º/s)
        usleep(100000);
        break;

    case 3:
        MPU6050_write(ACCEL_CONFIG, 0x10); // For mpu6050_accelerometer
sensitivity 2 (±8 g)
        usleep(100000);
        MPU6050_write(GYRO_CONFIG, 0x10); // For mpu6050_gyro sensitivity
2 (±1000 º/s)
        usleep(100000);
        break;

    case 4:
        MPU6050_write(ACCEL_CONFIG, 0x18); // For mpu6050_accelerometer
```

```
sensitivity 3 (±16 g)
            usleep(100000);
            MPU6050_write(GYRO_CONFIG, 0x18); // For mpu6050_gyro sensitivity
3 (±2000 º/s)
            usleep(100000);
            break;

        default:
            printf("enter the correct choice\n");
            break;
    }
}

//Here MPU6050_read_accel function will retrive the data from the registers of
accelerometer
void MPU6050_read_accel(uint8_t baseadd, int16_t *accdata){

    char buf;
    buf=baseadd;
    if(write(fd,&buf,1)<0){
        printf("error in the write\n");
    }

    int8_t rawdata[6];
```

```
    if(read(fd,rawdata,6)<0){
        printf("error in reading the raw data\n");
        return ;
    }

    accdata[0]= (rawdata[0]<<8 | rawdata[1]);
    accdata[1]= (rawdata[2]<<8 | rawdata[3]);
    accdata[2]= (rawdata[4]<<8 | rawdata[5]);
}

//The MPU6050_read_accel function will retrive the data from the registers of the gyroscope
void MPU6050_read_gyro(uint8_t baseadd, int16_t *gyrodata){

    char buf;
    buf=baseadd;
    if(write(fd,&buf,1)<0){
        printf("error in the write\n");
    }

    int8_t rawdata[6];
    if(read(fd,rawdata,6)<0){
        printf("error in reading the raw data\n");
        return ;
```

```
    }

    gyrodata[0]= (rawdata[0]<<8 | rawdata[1]);
    gyrodata[1]= (rawdata[2]<<8 | rawdata[3]);
    gyrodata[2]= (rawdata[4]<<8 | rawdata[5]);
}

int main(){
    //to store the of accelerometer and gyro
    int16_t accdata[3], gyrodata[3];
    double accx,accy,accz,gyrox,gyroy,gyroz;

    //to open the node which is associated with the mpu6050 to communicate
    fd=open("/dev/i2c-2", O_RDWR);
    if(fd<0){
        printf("error in opening the file\n");
        return -1;
    }

    //to set mpu6050 as i2c slave
    if (ioctl(fd, I2C_SLAVE, MPU6050_ADDR) < 0){
        printf("error in ioctl setting slave\n");
        return -1;
    }
```

```
    MPU6050_init ();  //initializing the mpu6050

/*
  ------------------------------------TO FETCH RAW DATA----------------------------
----------

    //infinite loop to fetch the raw data from the
    while (1)
    {
        MPU6050_read_accel(ACCEL_XOUT_H, accdata);
        MPU6050_read_gyro(GYRO_XOUT_H, gyrodata);

        printf("Accelerometer: X=%d, Y=%d, Z=%d\n", accdata[0], accdata[1],
accdata[2]);
        printf("Gyroscope: X=%d, Y=%d, Z=%d\n", gyrodata[0], gyrodata[1],
gyrodata[2]);

        usleep(500000); //sleep for 0.5 sec
        }
*/

   switch(choice){

        case 1:
```

```
    while(1){
        MPU6050_read_accel(ACCEL_XOUT_H, accdata);
        MPU6050_read_gyro(GYRO_XOUT_H, gyrodata);

        /Convert acc raw values in to 'g' values/
        accx = (double) accdata[0]/ACC_FS_SENSITIVITY_0;
        accy = (double) accdata[1]/ACC_FS_SENSITIVITY_0;
        accz = (double) accdata[2]/ACC_FS_SENSITIVITY_0;

        /* Convert gyro raw values in to  "°/s" (deg/seconds) */
        gyrox = (double) gyrodata[0]/GYR_FS_SENSITIVITY_0;
        gyroy = (double) gyrodata[1]/GYR_FS_SENSITIVITY_0;
        gyroz = (double) gyrodata[2]/GYR_FS_SENSITIVITY_0;

        /* print the 'g' and '°/s' values */
        printf("Acc(g)=> X:%.3f Y:%.3f Z:%.3f\n",accx,accy,accz);
        printf("gyro(dps)=> X:%.3f Y:%.3f Z:%.3f\n",gyrox,gyroy,gyroz);

        usleep(500000); //sleep for 0.5 sec
    }
    break;

case 2:
    while(1){
```

```
        MPU6050_read_accel(ACCEL_XOUT_H, accdata);
        MPU6050_read_gyro(GYRO_XOUT_H, gyrodata);

        /Convert acc raw values in to 'g' values/
        accx = (double) accdata[0]/ACC_FS_SENSITIVITY_1;
        accy = (double) accdata[1]/ACC_FS_SENSITIVITY_1;
        accz = (double) accdata[2]/ACC_FS_SENSITIVITY_1;

        /* Convert gyro raw values in to  "°/s" (deg/seconds) */
        gyrox = (double) gyrodata[0]/GYR_FS_SENSITIVITY_1;
        gyroy = (double) gyrodata[1]/GYR_FS_SENSITIVITY_1;
        gyroz = (double) gyrodata[2]/GYR_FS_SENSITIVITY_1;

        /* print the 'g' and '°/s' values */
        printf("Acc(g)=> X:%.3f Y:%.3f Z:%.3f\n",accx,accy,accz);
        printf("gyro(dps)=> X:%.3f Y:%.3f Z:%.3f\n",gyrox,gyroy,gyroz);

        usleep(500000); //sleep for 0.5 sec

    }
    break;

case 3:
    while(1){
```

```
        MPU6050_read_accel(ACCEL_XOUT_H, accdata);
        MPU6050_read_gyro(GYRO_XOUT_H, gyrodata);

        /Convert acc raw values in to 'g' values/
        accx = (double) accdata[0]/ACC_FS_SENSITIVITY_2;
        accy = (double) accdata[1]/ACC_FS_SENSITIVITY_2;
        accz = (double) accdata[2]/ACC_FS_SENSITIVITY_2;

        /* Convert gyro raw values in to  "°/s" (deg/seconds) */
        gyrox = (double) gyrodata[0]/GYR_FS_SENSITIVITY_2;
        gyroy = (double) gyrodata[1]/GYR_FS_SENSITIVITY_2;
        gyroz = (double) gyrodata[2]/GYR_FS_SENSITIVITY_2;

        /* print the 'g' and '°/s' values */
        printf("Acc(g)=> X:%.3f Y:%.3f Z:%.3f\n",accx,accy,accz);
        printf("gyro(dps)=> X:%.3f Y:%.3f Z:%.3f\n",gyrox,gyroy,gyroz);

        usleep(500000); //sleep for 0.5 sec

    }
    break;

case 4:
    while(1){
```

```
            MPU6050_read_accel(ACCEL_XOUT_H, accdata);
            MPU6050_read_gyro(GYRO_XOUT_H, gyrodata);

            /Convert acc raw values in to 'g' values/
            accx = (double) accdata[0]/ACC_FS_SENSITIVITY_3;
            accy = (double) accdata[1]/ACC_FS_SENSITIVITY_3;
            accz = (double) accdata[2]/ACC_FS_SENSITIVITY_3;

            /* Convert gyro raw values in to  "°/s" (deg/seconds) */
            gyrox = (double) gyrodata[0]/GYR_FS_SENSITIVITY_3;
            gyroy = (double) gyrodata[1]/GYR_FS_SENSITIVITY_3;
            gyroz = (double) gyrodata[2]/GYR_FS_SENSITIVITY_3;

            /* print the 'g' and '°/s' values */
            printf("Acc(g)=> X:%.3f Y:%.3f Z:%.3f\n",accx,accy,accz);
            printf("gyro(dps)=> X:%.3f Y:%.3f Z:%.3f\n",gyrox,gyroy,gyroz);

            usleep(500000); //sleep for 0.5 sec

        }
        break;
    default:
        break;
}
```

return 0;

}

2.7 OPERATING SYSTEM

Our second part of the project is to build our own OS and run the executable generated on the target platform terminal (in beaglebone black). In order to build an embedded OS system, we need the following:

- Bootloader
- Kernel
- Root file system

U-BOOT

First, we need to download the bootloader file which is in a compresses format. Extract it and save it in a separate directory where we will also be extracting the kernel.

After extracting the u-boot give the following commands

cd </u-boot path>: go into the u-boot directory

cd configs: go into the configuration directory to find the **am335x_boneblack_vboot_defconfig file**

make clean: to remove/clear any previous files

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf am335x_boneblack_vboot_defconfig : this command will compile and builds the uboot

$ make menuconfig: used to do any changes in the uboot

When done, we will see some new files in the u-boot folder and the important files are:

MLO -It is the first stage bootloader

u-boot.img -second stage bootloader

u-boot.bin is the binary compiled U-Boot bootloader.

KERNEL

Download the kernel version which you want from kernel.org website. Extract the kernel file into your workspace directory. Go to the path where we downloaded the Linux kernel and give the following commands.

cd /linux/arch/arm/configs: to check the board support configuration and search for omap2plus_deconfig file.

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf omap2plus_deconfig file : to compile and build the linux kernel

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf -zImage -j4

When done, check in the /arch/arm/boot for the Kernel Image.

Image=> uncompressed kernel image

zImage=> compressed kernel image

DEVICE TREE BINARY

Give the following commands:

cd linux/arch/arm/boot/dts : check for the am335x-boneblack.dts in this path

$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf am335x-boneblack.dtb : to compile and build the device tree binary.

ROOTFS

Follow the given commands:

mkdir rootfs: create a rootfs directory

cd rootfs: go inside the rootfs directory

mkdir dev proc sys etc sbin lib mnt usr usr/bin usr/sbin: to create subdirectories required in the rootfs

Populate the bin,sbin usr/bin,usr/sbin using **busybox**

Download the **busy box** source code in the workspace directory.

make menuconfig: to do the required settings in the configuration

make CROSS_COMPILE=arm-linux-gnueabihf CONFIG_PREFIX=<path-to-rootfs> install

Next, we need to populate the /etc directory of ROOTFS. In the /etc file we need to create and write required commands in the inittab, init.d/rcS, profile and passwd files. After the /etc, we need to populate /dev file by creating device nodes for beagle bone.

 # cd <path-to-rootfs>/dev

 # mknod console c 5 1

 # mknod null c 1 3

 # mknod ttyO0 c 204 64

We need to give the below command to compile and build modules and if required, need to be installed.

make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf modules

 # make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf INSTALL_MOD_PATH=<path-to-rootfs> modules_install

SD CARD PARTITION

To partition your SD card, we first need to insert the SD card into the card reader and connect it to the host system. We make use of **Gparted** application which can easily create disk partitions /SD card partitions. We just need to install the Gparted application on our system and open it. It requires authentication, so it will ask you the user password. Once entered, a new terminal opens which shows the main disk partition, but we should not modify anything on this because a small change can format all the data on the disk. We need to select the card reader which will be displayed with the name “sdb”. If there are any existing partitions, we may want to delete the partitions and create new partitions. To do the partitions, we need to unmount the partitions and then delete the partitions. We done click on the green correct button on top to complete the deletion process.

To create partitions, we need to right-click and click on new. A new terminal opens where we are allowed to set the size of the partition, the type of the file system and the label we want to give. We partition the SD card into two partitions and label them as BOOT and ROOTFS with BOOT being FAT32 and ROOTFS having EXT4 file system. When done, click on apply and it’ll be done.

When all the above-mentioned steps are completed, we need to add the u-boot, MLO,uEnv.txt and zImage into the BOOT partition, while copying the rootfs content into the ROOTFS partition of the SD Card. To understand the internal working or the booting process, it is better to connect UART to TTL converter to the beaglebone black board and insert the SD card into the board. Press the boot (S2) button before giving the power supply and open the minicom terminal beforehand. If everything is done correctly, we will see the u-boot being loaded followed by the kernel and the root files. When done with everything, it will ask to login in the final stage. In this way we can build our own customized embedded OS Image.

3. SCHEMATIC DIAGRAM

3.1 BLOCK DIAGRAM

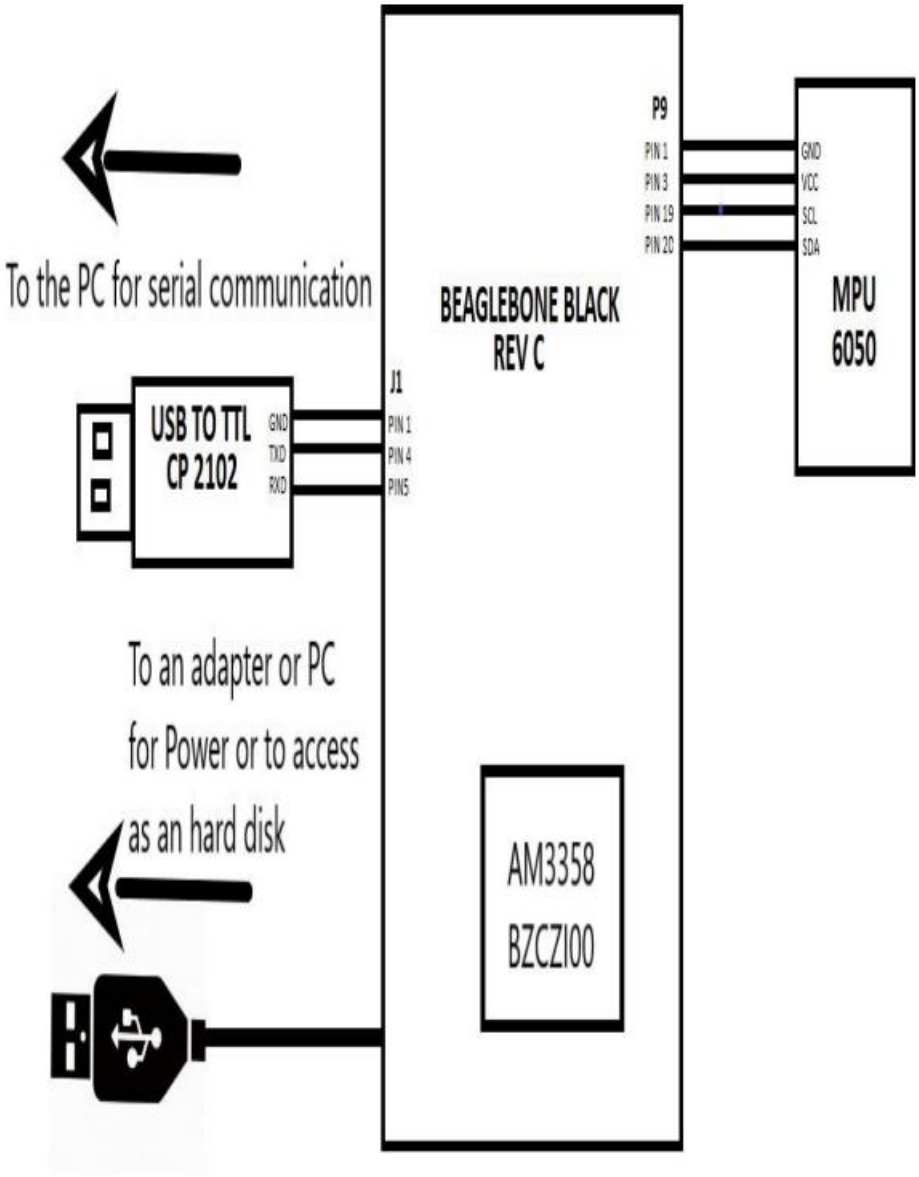

Figure 21 Block Diagram of the Interface

3.2 IMPLEMENTED SYSTEM

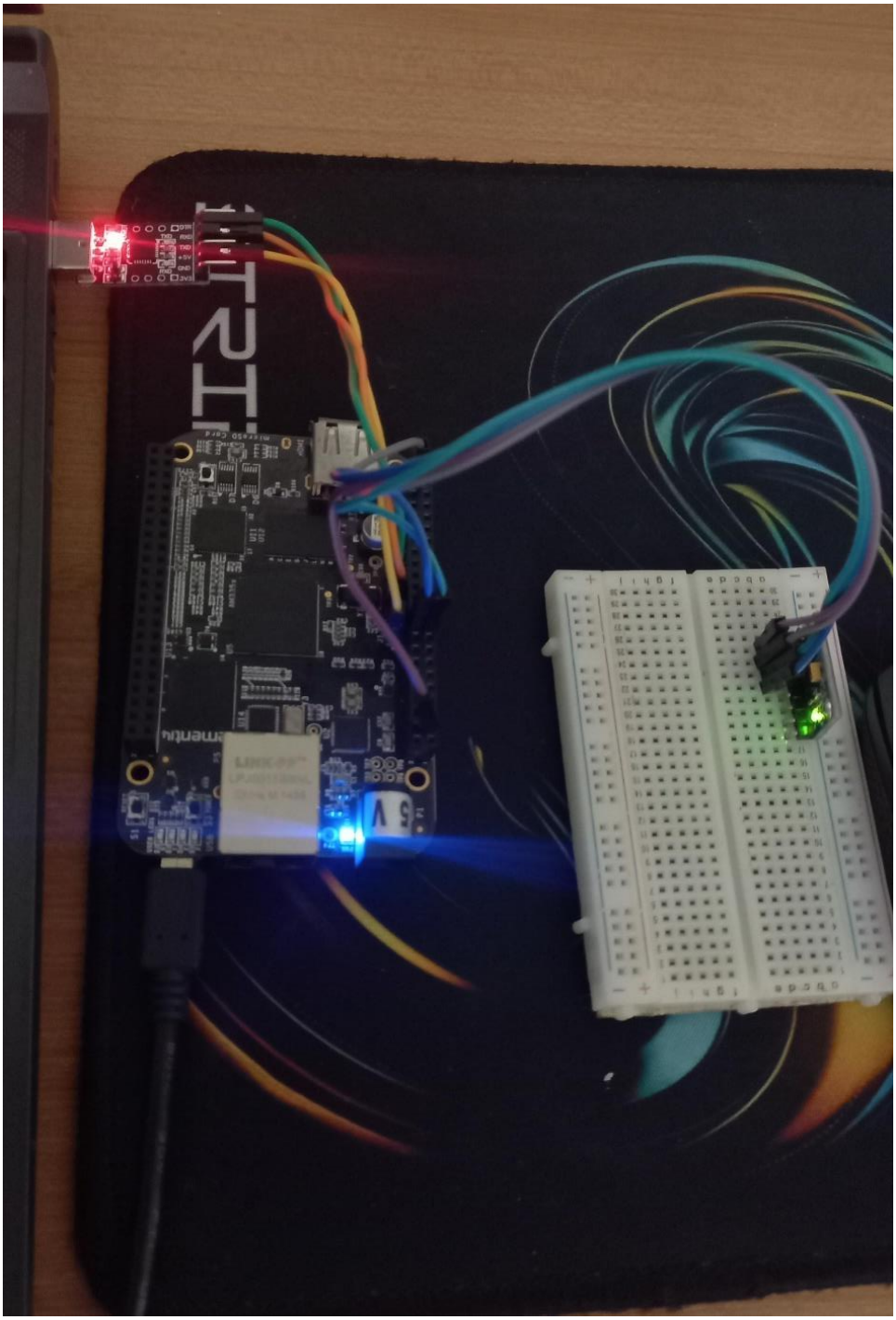

Figure 22 Project implementation set up.

4. FLOW OF PROJECT WORK

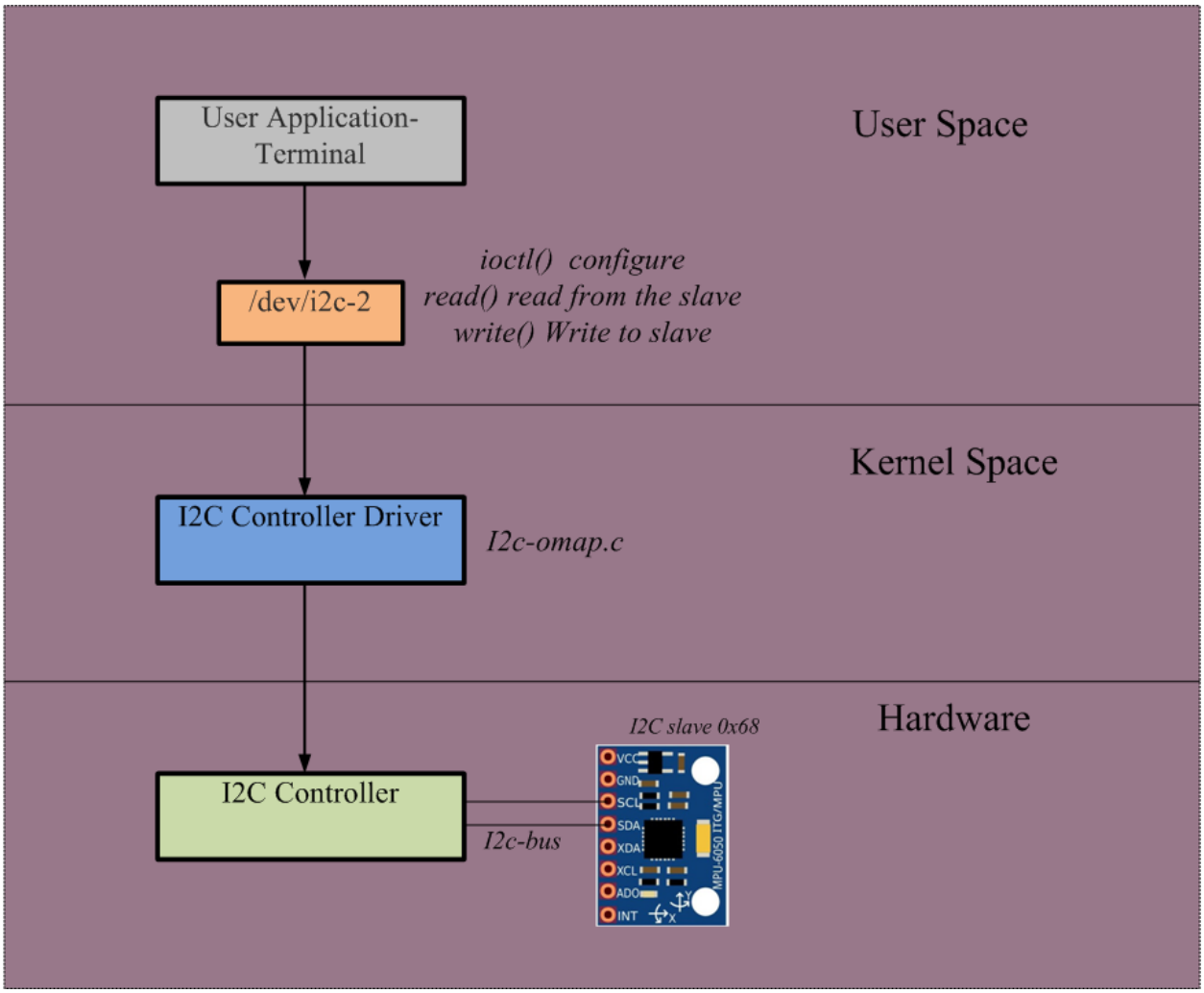

Figure 23 Workflow of the code

This diagram illustrates the sequence of steps involved in accessing each file to reach the desired output. It indicates the location of files, such as application files or device driver files, and the order in which they are accessed, opened, fetched, computed, and the output returned to the application program. The ioctl function is crucial for communication with the device driver.

5. LITERATURE SURVEY

In our literature review, we examined two research papers. The first paper, authored by **Jian Huang**, focuses on the design of an Angle Detection System using the MPU6050. The second paper, authored by **A. Yudhana, J. Rahmawan, and C. U. P. Negara**, explores the integration of flex sensors and MPU6050 sensors in a smart glove for sign language translation.

Paper 1, titled "Survey on Design of Angle Detection System Based on MPU6050" by **Jian Huang**, details the utilization of the MPU6050 to detect angles and outlines the hardware circuit design. The paper employs the Kalman filter algorithm in software programming to enhance angle detection accuracy by mitigating interference. Through experimentation, the authors demonstrate the system's capability to precisely measure both horizontal and vertical angles, showcasing its potential application in flight control systems.

Paper 2, titled "Flex sensors and MPU6050 sensors responses on smart glove for sign language translation" by **A. Yudhana, J. Rahmawan, and C. U. P. Negara,** discusses the integration of flex sensors and MPU6050 sensors into a smart glove for sign language translation. The flex sensors measure finger curvature, while the MPU6050 sensors detect hand movement. The authors utilize an Arduino Nano controller to gather sensor data, which is then displayed on a PC. They present data illustrating the correlation between finger gestures in sign language and sensor readings. The paper highlights the potential of additional accelerometer sensors in discerning unique gestures, such as certain letters, based on variations in slope values.

Overall, both papers contribute valuable insights into the application of MPU6050 sensors, either for angle detection or in conjunction with flex sensors for sign language translation, demonstrating practical implications in various fields.

6. CONCLUSION

Thus, the Tilt Sensor project utilizing the MPU6050 interfaced with Beaglebone Black was executed. This project holds numerous practical applications such as posture detection, adjusting mobile phone orientation, monitoring building health, and controlling Cruise/Aircraft mechanisms. The sensor underwent movement and rotation along all three axes, with the resultant raw and 'g' values observed and plotted to track changes along each axis.

7. REFERENCES

1. Yan Li, Shen Mingxia, Yao Wen, Lu Mingzhou,, et al. Method of gesture recognition for lactating sows based on MPU6050 sensor [J], Chinese Journal of agricultural machinery,.2015,46 (5): 279-285.

2. A Yudhana, J Rahmawan and C U P Negara, based on Flex sensors and MPU6050 sensors responses on smart glove for sign language translation, Indonesia at 2017 IOP Conf. Series: Materials Science and Engineering 403 (2018) 012032 doi:10.1088/1757-899X/403/1/012032

3. BBB_SRM Datasheet- https://cdn-hop.adafruit.com/datasheets/BBB_SRM.pdf

4.MPU6050 Datasheet- - https://pdf1.alldatasheet.com/datasheet-pdf/view/1132807/TDK/MPU-6050.html