**Beaglebone Black IoT Demo Board setup**

**TABLE OF CONTENTS**

**Chapter No. Description Page No.**

**CHAPTER 1 INTRODUCTION 01**

1.1 SPI master with signal path delay 01

compensation on PRU-ICSS01

1.2 Generation of Asynchronous pulse for

Hiperface DSl frame 02

1.3 CAN FD protocol Evaluation on PRU-ICSS 03

**CHAPTER 2 SPI MASTER WITH SIGNAL PATH DELAY**

**COMPENSATION ON PRU-ICSS 04**

2.1 Hardware and Software 04

2.1.1 AM437x IDK 04

2.1.2 AM437x sitara processor 05

2.1.3 PRU-ICSS 05

2.1.3.1 GPIO 06

2.1.3.2 28-bit serial shift register 06

2.1.4 Memory 07

**Chapter – 01**

**INTRODUCTION**

In this we developed a demo by connecting different resistive and capacitive sensors to the Beaglebone Black and get the data from those sensors and publishing the data to the AWS IoT cloud for analyzing and processing it for further future cases.

The detailed information on Software, Hardware, AWS IoT cloud and package installations are explained in further documentation.

Here final demo diagram with the AWS iot cloud image .

**CHAPTER – 02**

**SOFTWARE AND HARDWARE**

**2.1 List of Hardware and Software**

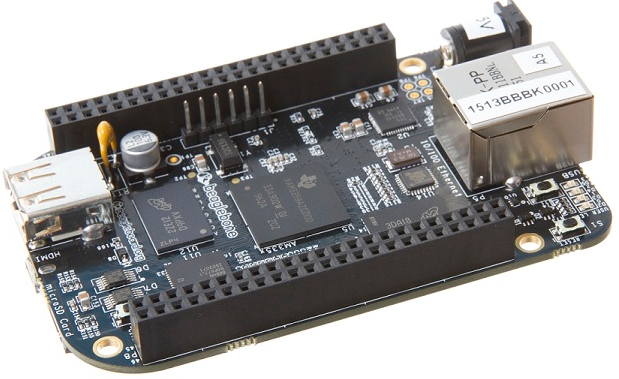
To develop this demo we used different softwares and hardwares which are listed below:

* Beaglebone Black Rev.C
* 4-port USB Hub
* Breadboard and Jumper wires
* Resistors and LED
* WiFi module with the TP-Link
* Servo motor
* Sensors(Temperature, Light, Force resistive and Distance measurement)
* Debian Weezy Operating system
* AWS IoT cloud
* MQTT client
* AWS IoT device SDK

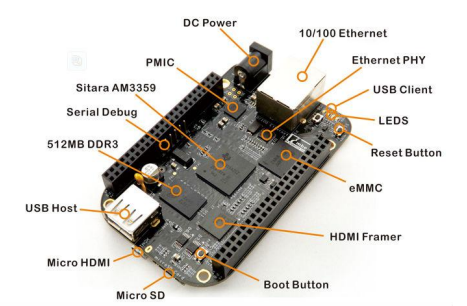
The programming languages used are Python, C and C++, Java script with node.js.

**2.1.1 Beaglebone Black Rev.C**

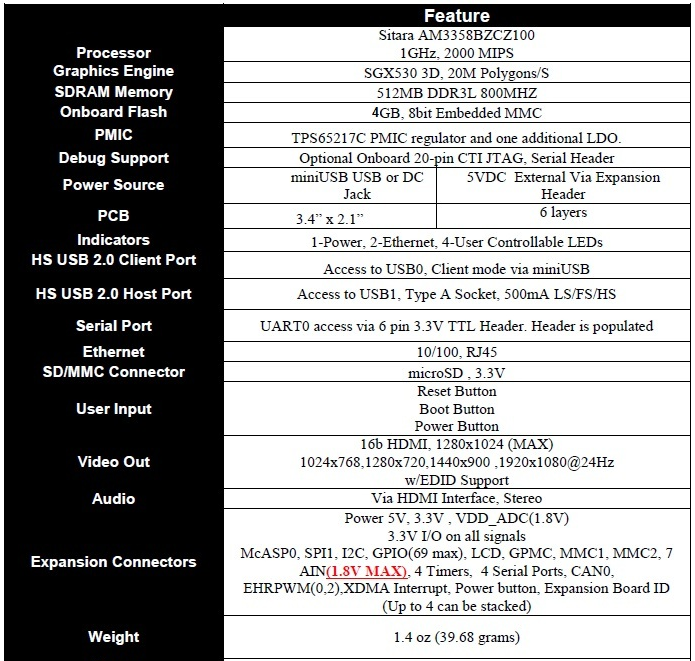
BeagleBone Black is a low-cost, community-supported development platform for developers with AM335x 1GHz ARM® Cortex-A8 processor, 512MB DDR3 RAM and 4GB 8-bit eMMC on-board flash storage . Boot Linux in under 10 seconds and get started on development in less than 5 minutes with just a single USB cable.



Beaglebone Black



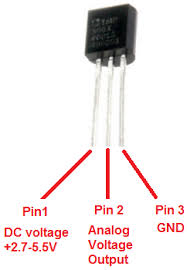
Beaglebone Black with the components



Features of Beaglebone Black

**2.1.2 Other Hardware**

We used 4-port USB hub which is connected to the BBB to connect the external devices like WiFi adapter, Mouse and Keyboard. Breadboard and jumpers to connect the sensors to the BBB. The WiFi adapter is Logilink 802.11n. The sensors are the Temperature sensors which is TMP36, Mini servo motor, Photoresistor as light sensor, Distance Measurement IR sensor, Round Force resistive sensor and some resistors and LED’s.



**CHAPTER – 03**

**INSTALLATIONS**

**3.1 Operating system Installation on BBB**

First step to start is with flashing operating system on the BBB. We flashed Debian Weezy and used as an operating system for developing this.

Step-by-step procedure for Flashing OS into the BBB:

* Download the latest software image from <http://beagleboard.org/latest-images> ().
* Download the latest software image from http://beagleboard.org/latest-images (Debian 4GB eMMC Flasher).
* Decompress the SD card .img file using 7-zip.
* Install the 4-Gb SD card into the SD card reader and connect it to the computer
* Using SD-Formatter format the SD card
* Using Win32 Disk manager write the .img file to the SD card (http://www.computerbild.de/download/Win32-Disk-Imager-5459603.html)
* After write successful window eject the SD card from the computer
* Insert SD card into (powered-down) board, hold down the USER/BOOT button and apply power, either by the USB cable or 5V adapter and hold down the button until the USRx LEDs start glowing.
* When the flashing is complete, all 4 USRx LEDs will be steady on or off. The latest Debian flasher images automatically power down the board upon completion. This can take up to 45 minutes.
* Power-down your board, remove the SD card and apply power again to be complete the flashing of OS on to the board.

Power down the board and connect the USB hub on to the board and connect Mouse, Keyboard and WiFi adapter to the hub, Connect the screen using micro HDMI to VGA connector(Use a external power supply to connect the micro HDMI to VGA connector otherwise the screen will not get connected to the board) and power up the board you will see the screen with the debian OS and you can use the external devices to communicate with it.

**3.2 Installing WiFi adapter**

First we need to see whether the adapter is connected or not, for this open the terminal and type

lsusb – you should see the wireless adapter name in the devices

Open the Interface file which is in ‘nano /etc/network/interfaces’ and search for the following lines, uncomment them and give the SSID and the password, in this case I connected the BBB to a TP-Link which is connected to a LAN connection because it cannot authenticate the WiFi which has to do an authentication as it doesn’t have a real time clock, for example we cannot connect to ‘reply-wpa’ WiFi which asks for authentication, so its better to connect to Open network and provide with the SSID and password.

# WiFi Example

auto wlan0

iface wlan0 inet dhcp

wpa-ssid "TP-LINK\_EEB89A"

wpa-psk "1FEEB89A"

Open the Wicd Network manager from the Start and Internet options and scan for the WiFi networks and go to the properties of the network or SSID you want to connect and provide the necessary username and passwords. Click on Automatically connect to this network so that on reboot it connects to the network automatically and Clcick on Connect button for connection.

Then type ‘iwconfig’ in the terminal to see whether the wlan0 is connected to the SSID you choosed. Then type ‘ping www.google.com’ to see the device is connected and have internet access.

**3.3 Installing GIT**

Most of the packages and the source codes are on git, so initially we need to install git and configure.

sudo apt-get install git

This is used for later installations and for git clone.

**3.4 Configuring Chromium Browser**

Initially we can use chromium browser from debian user but if you are using it as root privileges there will be an error. So to use chromium as root user we need to change the flags of the chromium file which is in

‘nano /etc/chromium/dafault’ and change the line CHROMIUM\_FLAGS = “--user -data-dir”

Then save, exit the file and launch the chromium browser.

**3.5 Installing Python and configuring GPIO, ADC, PWM, SPI and I2C interfaces**

To install the python on BBB we need to select the IO libraries for the ADC, GPIO, so in our case I used the Adafruit BBIO library. To install this Adafruit BBIO library and python is shown below.

sudo apt-get update

sudo apt-get install build-essential python-dev python-setuptools python-pip python-smbus -y

git clone git://github.com/adafruit/adafruit-beaglebone-io-python.git

cd adafruit-beaglebone-io-python

sudo python setup.py install

cd ..

import Adafruit\_BBIO.GPIO as GPIO

import Adafruit\_BBIO.PWM as PWM

import Adafruit\_BBIO.ADC as ADC

from Adafruit\_I2C import Adafruit\_I2C

from Adafruit\_BBIO.SPI import SPI

import Adafruit\_BBIO.UART as UART

By this we are done with the installation on python and can use the GPIOs, ADC etc.

**3.6 Configuring GPIOs and ADC for using with other languages**

**3.6.1 For C and C++ Languages**

In BBB the C and C++ compilers are built in with the Debian installation. For these languages to use GPIOs and ADC interfaces we need to use by the path of those pins like

‘/sys/devices/ocp.3/helper.15/AINx‘ where x is 0 to 6 and for ADC

‘/sys/class/leds/beaglebone\:green\:usr%d/brightness’ where %d is 0, 1 and 2 for user leds

‘/sys/class/gpio/’ for GPIO pins

**3.6.2 For bonescript and javascript(node.js)**

The other languages like bonescript and Javascript we need node.js to be installed. So initially we install node.js on BBB.

wget http://nodejs.org/dist/v0.10.5/node-v0.10.5.tar.gz

tar xzvf node-v0.10.5.tar.gz

cd node-v0.10.5

./configure --without-snapshot

Make

./node -e 'console.log("het werkt!");'

./node –v

make install

Thus we install node.js and we can write bonescript and javascript.

BoneScript is a Node.js library specifically optimized for the Beagle family and featuring familiar Arduino function calls, exported to the browser.

**3.7 Installation of Openssl and paho mqtt for python and C language**

The OpenSSL is a collaborative effort to develop a robust, commercial-grade, full-featured, and Open Source toolkit implementing the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols as well as a full-strength general purpose cryptography library.

wget https://www.openssl.org/source/openssl-1.0.2.tar.gz

tar zxf openssl-1.0.2.tar.gz

cd openssl-1.0.2/

ls

./config -DHAVE\_CRYPTODEV -DUSE\_CRYPTDEV\_DIGESTS

make

sudo make install

MQTT is a system, and a network protocol, for broadcasting and listening for information over networks. Designed as a complement to enterprise messaging systems, MQTT is a lightweight publish and subscribe protocol. The idea with what was called at the time, Message Queuing Telemetry Transport, to fill the gap between the numerous devices and applications that could produce data and the wider world of data consumers. The installation is as follows:

git clone git://git.eclipse.org/gitroot/paho/org.eclipse.paho.mqtt.python.git

cd org.eclipse.paho.mqtt.python

make

make install

and use ‘import paho.mqtt.client as mqtt’ in the programs.

**CHAPTER – 04**

**AWS IoT**

**4.1 Introduction**

AWS IoT enables secure, bi-directional communication between Internet-connected things (such as sensors, actuators, embedded devices, or smart appliances) and the AWS cloud over MQTT and HTTP. It enables you to connect devices to AWS Services and other devices, secure data and interactions, process and act upon device data, and enable applications to interact with devices even when they are offline. It provides a number of ways to interact with the service platform, using the MQTT and HTTP RESTful protocols, and across different language frameworks.

The C SDK can be compiled on Linux distributions using OpenSSL or mbedTLS libraries and the included MQTT library. Customers can also port this SDK to additional embedded platforms and incorporate custom TLS and MQTT libraries, if needed. A special version of the C SDK is provided for Arduino Yún. The Device SDK is available for JavaScript in the Node.js package ecosystem.

The aws-iot-device-sdk.js package allows developers to write JavaScript applications which access the AWS IoT Platform; it is intended for use in embedded devices which support Node.js.

**4.2 Installation of AWS IoT SDK**

**4.2.1 Installation of AWS IoT device SDK for Embedded C**

Create a directory to hold your application e.g. (mkdir /aws\_iot\_Embedded\_c/)

Change directory to aws\_iot\_Embedded\_c directory

Download the SDK to device from <https://github.com/aws/aws-iot-device-sdk-embedded-C> and place in the newly created directory

Expand the tarball (tar -xf ). This will create 4 directories:

aws\_iot\_src - the AWS IoT SDK source files

sample\_apps - the sample applications

aws\_mqtt\_embedded\_client\_lib - the aws MQTT client derived from Eclipse Paho Embedded C client

certs - TLS certificates directory

Change directory to sample\_apps. The following sample applications are included:

subscribe\_publish\_sample - a simple pub/sub MQTT example

shadow\_sample - a simple device shadow example using a connected window example

shadow\_sample\_console\_echo - a sample to work with the AWS IoT Console interactive guide

For each sample:

Explore the example. It connects to AWS IoT platform using MQTT and demonstrates few actions that can be performed by the SDK

Build the example using make. (''make'')

Place device identity cert and private key in locations referenced in the example (certs/). Alternatively, you can run the sample application with the ''-c'' flag to point to a specific certificate directory.

Download certificate authority CA file from Symantec and place in location referenced in the example (certs/). Ensure the names of the cert files are the same as in the aws\_iot\_config.h file

Run sample application (./subscribe\_publish\_sample or ./shadow\_sample). The sample will print status messages to stdout.

**4.2.2 Installation of AWS-IoT device Node.js SDK**

The more information is on <https://github.com/aws/aws-iot-device-sdk-js> but the basic installation is shown here.

npm install aws-iot-device-sdk

git clone https://github.com/aws/aws-iot-device-sdk-js.git

cd aws-iot-device-sdk-js

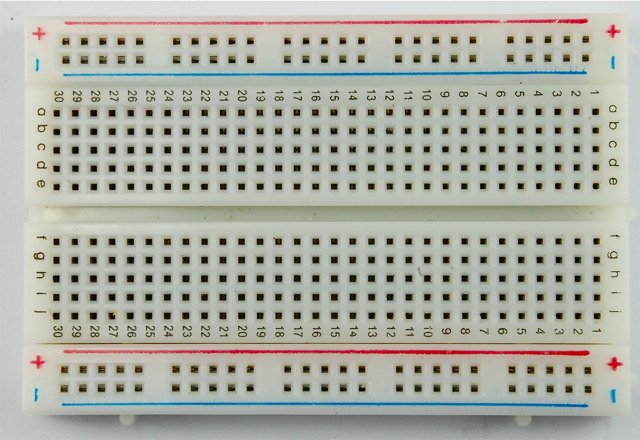
npm install

**CHAPTER – 05**

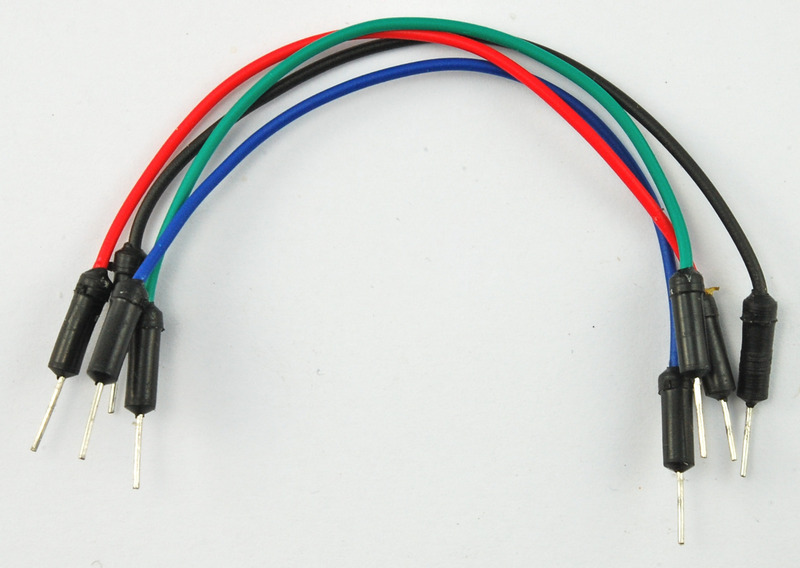
**CONNECTING SENSORS TO BEAGLEBONE BLACK**

**5.1 Requirements**

For connecting the sensors, initially we connected on the breadboard and later prototyped it to the main pcb board. Initial things required for connecting sensors are Breadboard, jumper wires and resistors.



Breadboard



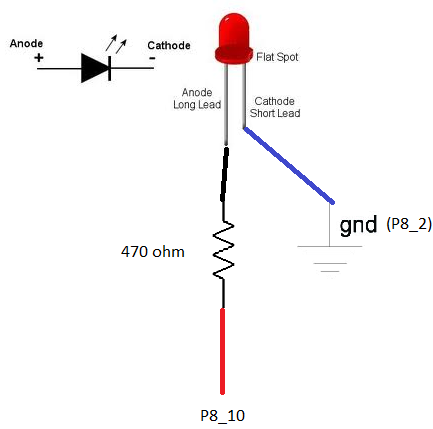
Jumper wires



Resistor

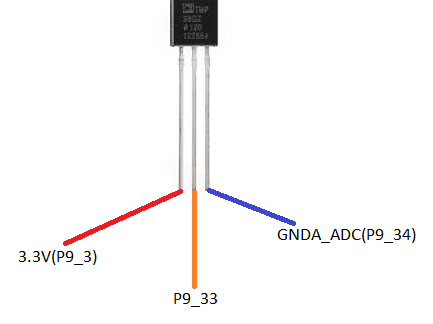
**5.2 Connecting an LED**

The resistor can be connected anyway around. For the LED, the longer (Anode) lead goes to the one end of resistor and the other end of resistor goes to P8\_10 pin. The Shorter (Cathode) lead goes to the P8\_2 pin which is ground. The connections are shown below.



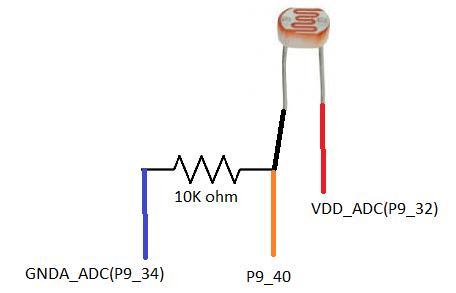
**5.3 Connecting a Temperature Sensor**

In this we used a TMP36 temperature sensor and three jumper wires blue, red and orange. The blue lead is connected from the GNDA\_ADC connection to the GND pin of the TMP36 temperature sensor. The red lead is connected from pin 3 of the other connector (3.3V) to the positive supply pin of the TMP36 and the orange lead from Analog voltage output to pin P9.40 of BBB which is shown below



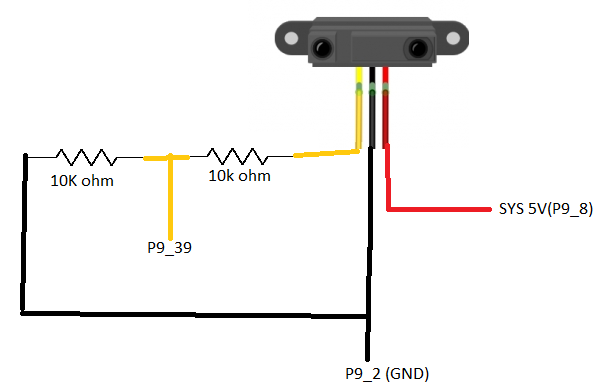
**5.4 Connecting a Light Sensor**

The photoresistor and the resistor can be placed either way around. The orange lead to pin P9.40, which is also analog input 1 (AIN1). Connect the blue lead from pin 34 of P9 to the bottom of the resistor and the red lead from pin 32 of P9 to the top connection of the photoresistor. The connections are shown below.



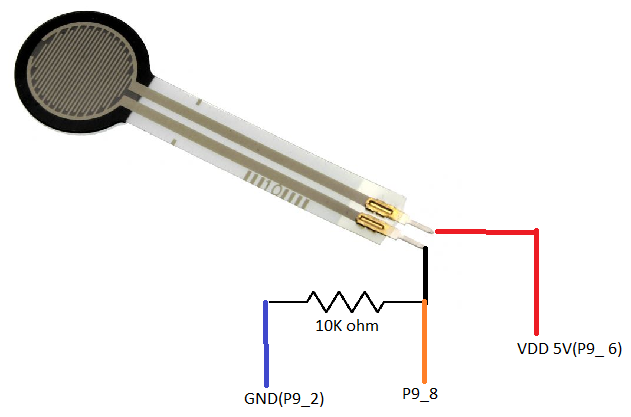
**5.5 Connecting a Distance Measurement sensor**

The Distance measurement sensor has 3 wires which are Red, Black and Yellow in color. The red wire is connected to the P9\_8 which is SYS 5V, the black wire is connected to ground on P9 which is 2nd pin and the yellow wire is connected to the one end of series 10K ohm resistor and the other end to ground and the we connect P9\_39 pin to the middle of resistors. The connections are shown below.



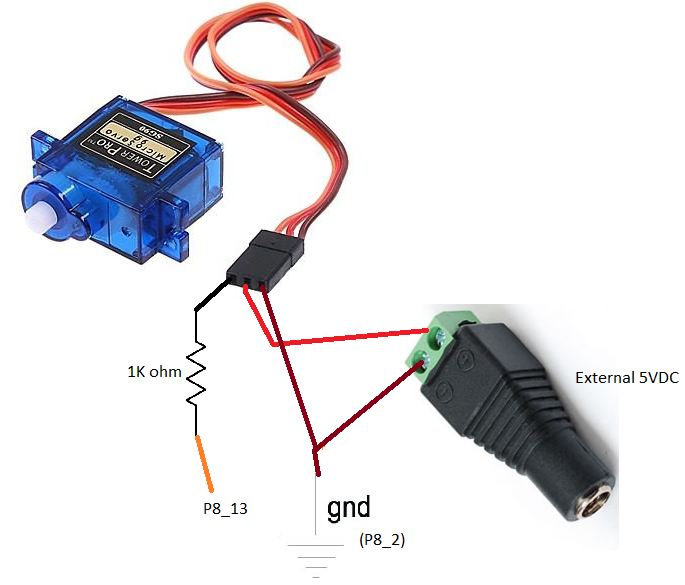
**5.6 Connecting a Force Resistive Sensor**

The Force resistive sensor has 2 ends and can be connected any way around. Use red wire to connect to P9\_6 which is 5V VDD to one end of force resistive sensor ,The other end is connected to one end of 10K ohm resistor which in series goes to P9\_8 and the other end of resistor goes to ground i.e. P9\_2 pin. The connections are shown below.



**5.7 Connecting a Micro-Servo motor**

Use an external 5VDC power supply jack and connect +ve to red wire of motor and –ve to the groung on BBB. We will use pin P8\_13 as the PWM output to control the servo. Connect orange lead from P8\_13 to the one end of resistor and other end to orange wire of motor. Connect brown wire to ground of BBB i.e. P8\_2 pin. The connections are shown below.



**5.8 Prototyping the sensors and Beaglebone black**

**CHAPTER – 06**

**PROGRAMMING THE SENSORS**

**6.1 Programming Sensors using Python language**

The initial language used in this demo is python to test the sensors and get the data from them. It is easy to test the sensors using python because it has the predefined libraries. The basic python programming looks like

#!/usr/bin/python #this is to make the file executable

import Adafruit\_BBIO.ADC(GPIO) as ADC(GPIO) #importing ADC, GPIO libraries

import time

sensor\_pin = 'Px\_y' #x is the port number which is 8 or 9 and y is pin number from 1 to 46

ADC.setup() #setup ADC

GPIO.setup("Px\_y", GPIO.OUT(IN)) #GPIO setup and the mode setup for that pin

while True:

reading = ADC.read(sensor\_pin) #reading sensor pin

GPIO.output("Px\_y", GPIO.HIGH(LOW)) #Making GPIO High or Low

Function and code implementation

**6.2 Example codes for sensors using Python**

The codes for the sensors used in this demo are shown below.

**LED**

<https://github.com/karthikbnd/BBB-codes/blob/master/led1.py>

**Temperature sensor**

<https://github.com/karthikbnd/BBB-codes/blob/master/temperature1.py>

**Light sensor**

<https://github.com/karthikbnd/BBB-codes/blob/master/light1.py>

**Distance measurement sensor**

<https://github.com/karthikbnd/BBB-codes/blob/master/distance1.py>

**Force resistive sensor**

<https://github.com/karthikbnd/BBB-codes/blob/master/fsr1.py>

**Servo motor**

<https://github.com/karthikbnd/BBB-codes/blob/master/servo1.py>

The program used for all sensors is shown below and which writes the output data to a file.

<https://github.com/karthikbnd/BBB-codes/blob/master/project1.py>

#!/usr/bin/python

import Adafruit\_BBIO.ADC as ADC

import Adafruit\_BBIO.GPIO as GPIO

import time

import math

my\_file = open("Result.txt","w") # Write the results to a txt file

sensor\_pin = 'P8\_10' # LED pin

sensor\_pin1 = 'P9\_33' # Temperature sensor pin

sensor\_pin2 = 'P9\_37' # IR sensor pin

sensor\_pin3 = 'P9\_38' # Force resistive sensor pin

sensor\_pin4 = 'P9\_40' # Light sensor pin

ADC.setup()

GPIO.setup("P8\_10", GPIO.OUT) #led OUT

my\_file.write("Temperature\_C \t Temperature\_F \t Distance(cm) \t Force(v) \t Light(v) \n")

while True:

GPIO.output("P8\_10", GPIO.HIGH)

# Reading sensor values

reading1 = ADC.read(sensor\_pin1)

reading2 = ADC.read(sensor\_pin2)

reading3 = ADC.read(sensor\_pin3)

reading4 = ADC.read(sensor\_pin4)

# Calculation of temperature

millivolts = (reading1 / 4096)\* 1800

temp\_c = (millivolts - 500) / 10

temp\_f = (temp\_c \* 9/5) + 32

c = str(temp\_c)

f = str(temp\_f)

print('mv=%d C=%d F=%d' % (millivolts, temp\_c, temp\_f))

# Calculations of IR sensor

voltage = reading2 \* 1.65 #values from 0 to 1.65V

distance = 13.93 \* pow(voltage, -1.15)

d = str(distance)

if distance > 80:

print("Can't measure more than 80cm!")

else:

print("The reading, voltage and distance (in cm) are " + str(reading2), str(voltage), str(distance))

# Calculations of FSR sensor

voltf = reading3 \* 1.800

f=str(voltf)

print('reading\_FSR=%f \t Volt=%f' % (reading3, voltf))

# Calculations of Light sensor

voltl = reading4 \* 1.800

l=str(voltl)

print('reading\_Light=%f \t Volt=%f' % (reading4, voltl))

my\_file.write(c + '\t' + f + '\t' + d + '\t' + f + '\t' + l + '\n')

time.sleep(0.5)

Using python language its easy to test the sensors but when we move further to the paho mqtt its not possible to use because the paho mqtt uses the python 3.4.3 and later versions but the Adafruit library of beagelbone black uses 2.7.2 version. So it is not possible to use the paho mqtt. So now we are moving to C language.

**6.3 Programming sensors using C language and using AWS IoT device SDK for Embedded C**

As we are unable to use python adafruit library with paho mqtt, now we are going to use C language. For C language we use the AWS IoT device SDK for Embedded C from Amazon web services for publishing and subscribing the data on the cloud. The programs written for the sensors are shown below.

**Temperature Sensor**

<https://github.com/karthikbnd/BBB-codes/blob/master/ctemp.c>

**Light Sensor**

<https://github.com/karthikbnd/BBB-codes/blob/master/clight.c>

**Distance measurement Sensor**

<https://github.com/karthikbnd/BBB-codes/blob/master/cdistance.c>

**Force resistive Sensor**

<https://github.com/karthikbnd/BBB-codes/blob/master/cfsr.c>

The codes are used for different individual sensors to connect with them and get the data from them. Then we installed the AWS IoT device Embedded C SDK which as the mqtt client and can publish the data. But when we execute the file and try to publish there is an error regarding the VLS TS error which is not supported by the BBB. So then we tried another language which is node.js using bonescript.

**6.4 Programming sensors using node.js, bonescript and using AWS IoT device SDK for js (JavaScript)**

As we are unable to use python adafruit library with paho mqtt and C language, now we are going to use javascript and bonescript with node.js. For this we use the AWS IoT device SDK for node.js from Amazon web services for publishing and subscribing the data on the cloud. Finally we tested with this and it works. For writing the bonescript we should create a directory and add the Bonescript library(downloaded from <https://github.com/jadonk/bonescript>), Restify(npm install restify) and downloaded AWS IoT SDK into that directory and start writing or modifying the programs in the same directory. You can also download everything from my github folder which is named ‘temperature’ <https://github.com/karthikbnd/BBB-codes/tree/master/temperature>. The program written for every sensor is shown below:

<https://github.com/karthikbnd/BBB-codes/blob/master/temperature/temp2AWSv1.4.js>

//BeagleBone NodeJs library

var b = require('bonescript');

//AWS IoT NodeJs library

var awsIot = require('aws-iot-device-sdk');

//Setting IoT device parameters

var device = awsIot.device({

keyPath: '/root/awsCerts/private.pem.key',

certPath: '/root/awsCerts/certificate.pem.crt',

caPath: '/root/awsCerts/root-CA.crt',

clientId: 'BeagleBone',

region: 'eu-west-1'

});

//variables for sensor pins

var tempPin1 = 'P9\_33'; //temperature sensor

var tempPin2 = 'P9\_40'; //light sensor

var tempPin3 = 'P9\_37'; //IR sensor

var tempPin4 = 'P9\_38'; //FR sensor

var currentTemp = 0.0;

var currentLight = 0.0;

var currentDistance = 0.0;

var currentForce = 0.0;

var datetime;

b.analogRead(tempPin1, readTemperature) //read temperature from sensor

b.analogRead(tempPin2, readLight) //read Light from sensor

b.analogRead(tempPin3, readDistance) //read Distance from sensor

b.analogRead(tempPin4, readForce) //read Force from sensor

function temp2AWS() {

device

.on('connect', function() {

console.log('connect');

//device.subscribe('sdkTest/sub');

timeout = setInterval( function() {

b.analogRead(tempPin1, readTemperature) //read temperature from sensor

b.analogRead(tempPin2, readLight) //read Light from sensor

b.analogRead(tempPin3, readDistance) //read Distance from sensor

b.analogRead(tempPin4, readForce) //read Force from sensor

datetime = new Date().today() + " " + new Date().timeNow();

device.publish('sdkTest/DateTime', JSON.stringify({ Timestamp: datetime})); //publish date and time to AWS

device.publish('sdkTest/Temperature', JSON.stringify({ Temperature: currentTemp})); //publish temperature sensor data to AWS

device.publish('sdkTest/Light', JSON.stringify({ Light: currentLight})); //publish light sensor data to AWS

device.publish('sdkTest/Distance', JSON.stringify({ Distance: currentDistance})); //publish IR sensor data to AWS

device.publish('sdkTest/Force', JSON.stringify({ Force: currentForce})); //publish FR sensor data to AWS

device.publish('sdkTest/sub1', JSON.stringify({ Random\_Number: Math.random()})); //publish a random number to AWS

//console.log('publish message...');

}, 5000); // clip to minimum

});

}

temp2AWS();

// For temperature

function readTemperature (tRead) {

//console.log("Getting Temperature");

var x = (tRead.value\*3300);

var cel = (x-500)/100;

var fah = (cel \*9/5)+32;

currentTemp = cel;

}

// For light

function readLight (lRead) {

//console.log("Getting light");

var l = (lRead.value\*1800);

currentLight = l;

}

// For distance

function readDistance (dRead) {

//console.log("Getting distance");

var d = (dRead.value\*1.65);

var dis = 13.93 \* Math.pow(d,-1.15);

currentDistance = dis;

}

// For force

function readForce (fRead) {

//console.log("Getting force");

var f = (fRead.value\*1800);

currentForce = f;

}

// For todays date;

Date.prototype.today = function () {

return ((this.getDate() < 10)?"0":"") + this.getDate() +"/"+(((this.getMonth()+1) < 10)?"0":"") + (this.getMonth()+1) +"/"+ this.getFullYear();

}

// For the time now

Date.prototype.timeNow = function () {

return ((this.getHours() < 10)?"0":"") + this.getHours() +":"+ ((this.getMinutes() < 10)?"0":"") + this.getMinutes() +":"+ ((this.getSeconds() < 10)?"0":"") + this.getSeconds();

}