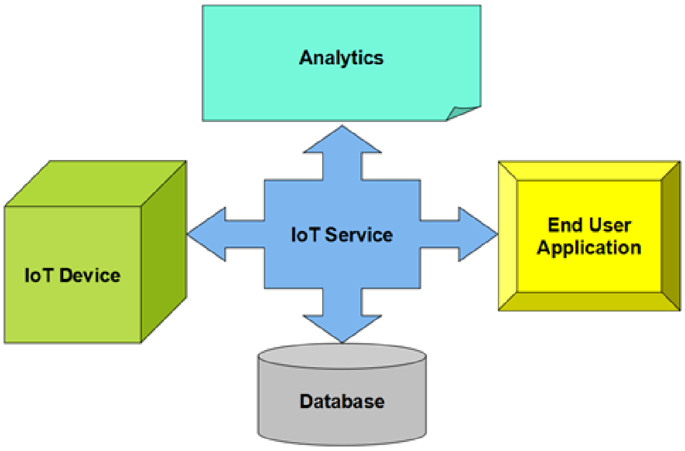
Laboratories for Day 3

Laboratory	Code
1. Sending data to Thingspeak online using wifi	Wee_TPH_ThingSpeak.ino and WeeESP8266 Library
2. Sending data to ThingSpeak private using GPRS	TPH_Volt_GPRS_TS.ino
3. Sleep mode	Serial_TPH_LP.ino and OnOff_test.ino
4. Real world exercise	**TPH_Volt_GPRS_TS_LP.ino**

1. ThingSpeak

1.1. Intro to ThingSpeak

The Internet of Things (IoT) is a system of ‘connected things’. The things generally comprise of an embedded operating system and an ability to communicate with the internet or with the neighboring things. One of the key elements of a generic IoT system that bridges the various ‘things’ is an IoT service. An interesting implication from the ‘things’ comprising the IoT systems is that the things by themselves cannot do anything. At a bare minimum, they should have an ability to connect to other ‘things’. But the real power of IoT is harnessed when the things connect to a ‘service’ either directly or via other ‘things’. In such systems, the service plays the role of an invisible manager by providing capabilities ranging from simple data collection and monitoring to complex data analytics. The below diagram illustrates where an IoT service fits in an IoT ecosystem:



One such IoT application platform that offers a wide variety of analysis, monitoring and counter-action capabilities is ‘ThingSpeak’. Let us consider ThingSpeak in detail.

What is ThingSpeak

ThingSpeak is a platform providing various services exclusively targeted for building IoT applications. It offers the capabilities of real-time data collection, visualizing the collected data in the form of charts, ability to create plugins and apps for collaborating with web services, social network and other APIs. We will consider each of these features in detail below.

The core element of ThingSpeak is a ‘ThingSpeak Channel’. A channel stores the data that we send to ThingSpeak and comprises of the below elements:

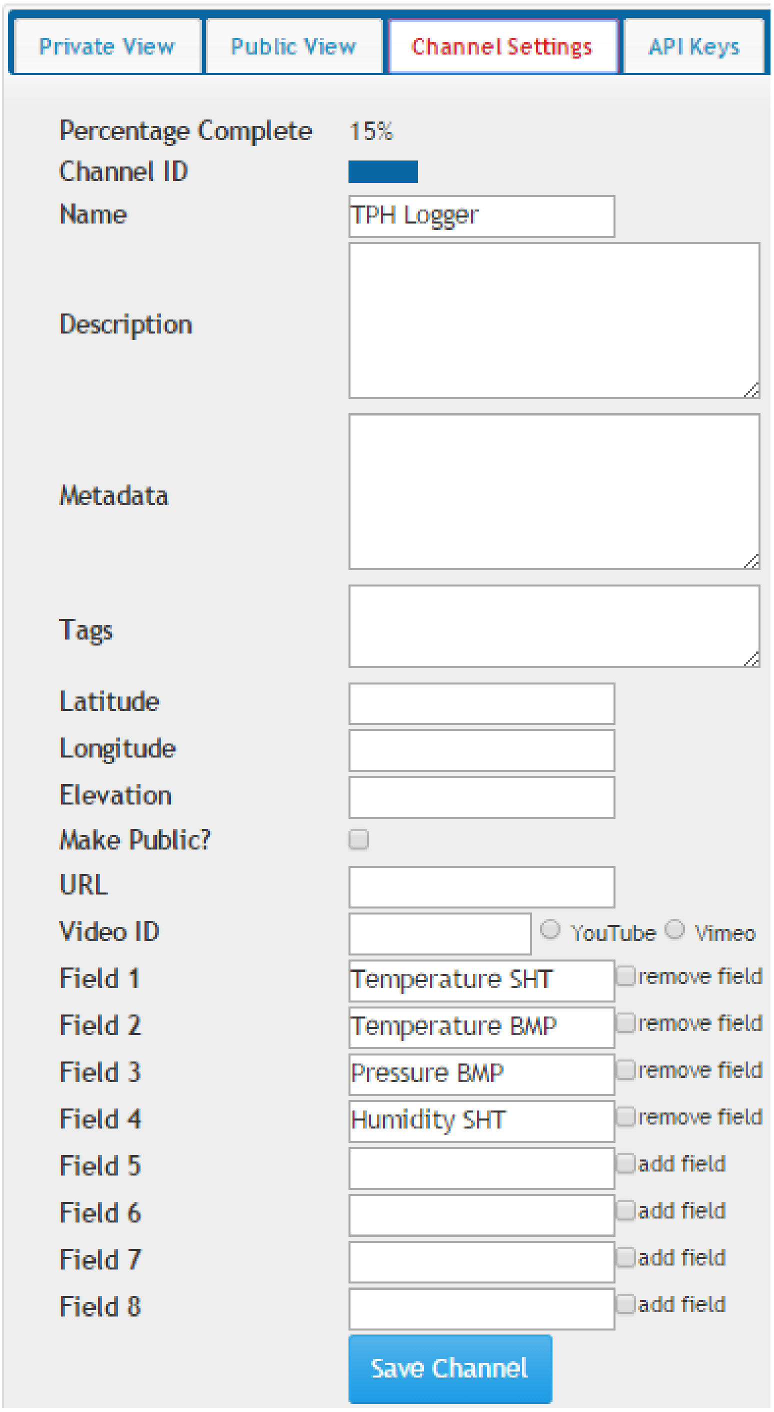
- 8 fields for storing data of any type - These can be used to store the data from a sensor or from an embedded device.
- 3 location fields - Can be used to store the latitude, longitude and the elevation. These are very useful for tracking a moving device.
- 1 status field - A short message to describe the data stored in the channel. To use ThingSpeak, we need to signup and create a channel. Once we have a channel, we can send the data, allow ThingSpeak to process it and also retrieve the same. Let us start exploring ThingSpeak by signing up and setting up a channel.

1.2. ThingSpeak Setup

Here are the steps required in order to get this example working with the ThingSpeak[1] website:

1. Create an account with ThingSpeak[2] (Sign-up)[3].
2. Create a new channel.
3. Copy the WRITE API KEY for your new channel (see the *API KEYS* tab).
4. Configure your new channel (see the *Channel Settings* tab).
    - You must to add four fields to your channel.
    - You should name the channel and each of the fields.
    - Make sure to save the new channel settings.
    - **Note:** The channel and field names are used for labelling the data in the charts shown on the private and public view tabs (see the image above). The names have no affect on the API and can be changed at any time. Here are the settings of the channel used to test this example:

[1] https://thingspeak.com/
[2] https://thingspeak.com/
[3] https://thingspeak.com/users/sign_up



Percentage Complete	15%
Channel ID	[redacted]
Name	TPH Logger
Description	
Metadata	
Tags	
Latitude	
Longitude	
Elevation	
Make Public?	☐
URL	
Video ID	○ YouTube ○ Vimeo
Field 1	Temperature SHT ☐ remove field
Field 2	Temperature BMP ☐ remove field
Field 3	Pressure BMP ☐ remove field
Field 4	Humidity SHT ☐ remove field
Field 5	☐ add field
Field 6	☐ add field
Field 7	☐ add field
Field 8	☐ add field

2. Wee WIFIBee Shield module

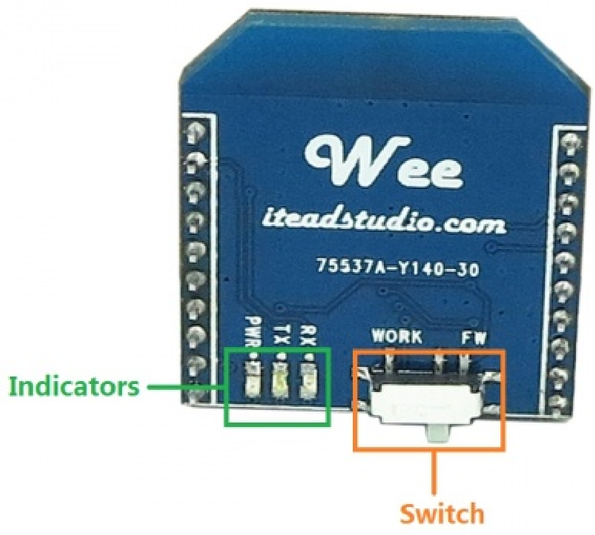
2.1. Overview

Wee is a WIFI module based on ESP8266 SoC. ESP8266 comes out of nowhere and has been taking by storm the IoT world. There are many hacking projects about it on the internet mainly because it is cheap, it costs around 5$.

So far, the most popular ESP8266 breakout version only has GPIO0 and GPIO2 routed to the header. Compared to it, Wee WIFI module is designed with a standard Bee interface and has more GPIOs available for developers. In a word, users can take full use of the utility of ESP8266 SoC by using Wee WIFI module in your projects.

2.2. Features

- Standard Bee interface
- indicators: TX, RX, PWR
- FW/Work Switch
- More GPIOs help developer take full use of the utility of ESP8266 SoC



Note Work/FW Switch. When burn firmware into ESP8266, you should ensure the switch is in FW mode. For the normal usage of this module, you should ensure the switch is in work mode.

2.3. Example

Connect Wee and TPH as follow

Wee wifi device is configured using AT commands. So, through the code wee is configured as follow:

Sending AT Wee answers OK if device is operative.

Sending AT+CWMODE=1 WiFi STA mode 1 for client, 2 AP, 3 it is both client and AP Wee answers.

AT+CIPMUX=0 Set Single connection.

AT+CWJAP="SSID","Password" Join accespoint.

AT+CIFSR Wee answer IP address.

Once Wee is connected to the access point we need to open a TCP connection with the following AT commands:

AT+CIPSTART="TCP","IP_Server",port

Once TCP connection is open we send data using,

AT+CIPSEND=data.length and we receive a '>', after thet we send data to server

Finally, we colse TCP connection

AT+CIPCLOSE

2.4. Sketch Code

```
#include <Wire.h>

//SODAQ Mbili libraries
#include <Sodaq_BMP085.h>
#include <Sodaq_SHT2x.h>
#include <Sodaq_DS3231.h>

//Node number
#define DEVICE_NUM "1"

//Data header
#define DATA_HEADER "Number,TempSHT21, TempBMP, PressureBMP, HumiditySHT21, Voltage"

//TPH BMP sensor
Sodaq_BMP085 bmp;

//*-- IoT Information
#define SSID   "WEEAPRPI2"
#define PASS   "weeaprpi2"
#define IP     "192.168.4.1"

//These constants are used for reading the battery voltage
#define ADC_AREF 3.3
#define BATVOLTPIN A6
#define BATVOLT_R1 4.7
#define BATVOLT_R2 10

void setup() {
  //start Serial
  Serial.begin(9600);
  //Start wifi Serial
  Serial1.begin(9600);
  //Initialise sensors
  setupSensors();

  //test Wifi response if OK connect it
  Serial1.println("AT");
  delay(2000);
  if(Serial1.find("OK"))
  {
    Serial.println("Wee OK. Data ready to be sent!");
    connectWiFi();
    delay(1000);
    checkIP();
  }
  else
  {
    Serial.println("Wee not responding to AT command");
  }
}

void loop() {
  String Temp1=String(SHT2x.GetTemperature());
  String Temp2=String(bmp.readTemperature());
  String Press=String(bmp.readPressure() / 100);
  String Hum=String(SHT2x.GetHumidity());

  //Read the voltage
  int mv = getRealBatteryVoltage() * 1000.0;
  String Smv=String(mv);

  String data= DEVICE_NUM ",";
  data += String(Temp1)  + ",";
  data += String(Temp2) + ",";
  data += String(Press)  + ",";
  data += String(Hum) + ",";
  data += String(Smv);

  TCPconn();

  //Echo the data header to the serial connection
  Serial.println(DATA_HEADER);
  Serial.println(data);

  TCPsend(data);
}

void setupSensors()
```

```
{
  //Initialise the wire protocol for the TPH sensors
  Wire.begin();

  //Initialise the TPH BMP sensor
  bmp.begin();

  //Initialise the DS3231 RTC
  rtc.begin();
}

//----- Connect to server using TCP connection
boolean TCPconn()
{
  //Connect to Server
  String cmd = "AT+CIPSTART=\"TCP\",\"";// Setup TCP connection
  cmd += IP;
  cmd += "\",333";
  Serial1.println(cmd);
  delay(2000);
  if( Serial1.find("Error"))
  {
    Serial.print( "Connection to Server failed" );
    return false;
  }else
  {
    Serial.println( "Connected to server" );
    return true;
  }
}

void TCPsend(String dat)
{
  /*Serial.print( "AT+CIPSEND=" );
  Serial.println(dat.length());*/
  Serial1.print("AT+CIPSEND=");
  Serial1.println(dat.length());
  //if > then post GET message
  if(Serial1.find( ">" ) )
  {
    Serial.print(">");
    Serial.println(dat);
    //Serial1.print("1");
    Serial1.print(dat);
    //check response
    if( Serial1.find("SEND OK") )
    {
      Serial.println( "Post to Server OK" );
    }
    else
    {
      Serial.println( "Post to Server Error" );
    }
  }

  //close connection
  Serial1.println( "AT+CIPCLOSE" );//close TCP connection
  if( Serial1.find("OK") )
  {
    Serial.println( "Closed connection: OK" );
  }
  else
  {
    Serial.println( "Closed connection Error" );
  }
  delay(10000); //
}

boolean connectWiFi()
{
  Serial1.println("AT+CWMODE=1");//WiFi STA mode - if '3' it is both client and AP
  delay(1000);
  Serial1.println("AT+CIPMUX=0");// Set Single connection
  delay(1000);
  //Connect to Router with AT+CWJAP="SSID","Password";
  // Check if connected with AT+CWJAP?
  String cmd="AT+CWJAP=\""; // Join accespoint
  cmd+=SSID;
  cmd+="\",\"";
  cmd+=PASS;
  cmd+="\"";
  Serial1.println(cmd);
  delay(1000);
  if(Serial1.find("ERROR"))
  {
    Serial.println("Connection to SSID ERROR");
    return false;
  }
  else
  {
    Serial.println("Connected to SSID");
    return true;
  }
}
```

```
void checkIP()
{
  Serial1.println("AT+CIFSR"); //ip address
  if (Serial1.available())
  {
      String ip = Serial1.readString();
      Serial.print("IP:");
      Serial.println(ip);
  }
}
float getRealBatteryVoltage()
{
  uint16_t batteryVoltage = analogRead(BATVOLTPIN);
  return (ADC_AREF / 1023.0) * (BATVOLT_R1 + BATVOLT_R2) / BATVOLT_R2 *
 batteryVoltage;
}
```

3. How To Install a ThingSpeak Server in a private way.

thingspeak.com[4] is a platform for The Internet of Things. If the free service is not suitable for your application (for example, your devices update more than once every 15 seconds), you might opt to install your own server.

The ThingSpeak source code is open-source and hosted on GitHub[5], but you may not find the installation as easy as "git clone". The platform is built using Ruby on Rails[6], and getting up to speed with Ruby, Gems, Rails, and the permutations and combinations of dependencies and package mangers may be more than you are willing to tackle just so you can do something silly like send a Tweet when the temperature in your living room gets too hot.

From https://github.com/iobridge/thingspeak reviewed steps to follow to install ThingSpeak server on a single board computer, Rpi, BBB or Alix

3.1. Reviewed steps to follow (remember not to put MYSQL password for ROOT)

```
sudo apt-get upgrade
sudo apt-get -y install build-essential mysql-server mysql-client libmysqlclient-dev
 libxml2-dev libxslt-dev git-core curl rubygems
```

[4] http://thingspeak.com/
[5] http://github.com/iobridge/ThingSpeak
[6] http://rubyonrails.org/

```
gpg --keyserver hkp://keys.gnupg.net --recv-keys
 409B6B1796C275462A1703113804BB82D39DC0E3
\curl -sSL https://get.rvm.io | bash -s stable
source /etc/profile.d/rvm.sh
rvm install 2.1
```

3.2. It takes time!

```
git clone https://github.com/iobridge/thingspeak.git
cd thingspeak
bundle install
```

3.3. It takes time again!

```
cp config/database.yml.example config/database.yml
rake db:create
rake db:schema:load
rails server webrick
```

If that's still too much for you, you can copy-and-paste this three-line version:

```
wget http://goo.gl/wS4hBf -O thingspeak-install.sh
chmod +x thingspeak-install.sh
./thingspeak-install.sh
```

During the installation you'll be asked to set a new root password for MySQL (if not already installed) and again when the ThingSpeak databases and tables are created.

DO NOT PUT A PASSWORD!!

When the installation is complete you can access your ThingSpeak server at http://XX.XX.XX.XX:3000. To shut it down, press Ctrl-C. To launch it again in the future, simply run "rails server".

3.4. Thingspeak server start automatically on boot

Create /home/user/runthingspeak.sh (replace user by your user name)

```
cat runthingspeak.sh
#!/bin/bash
cd /home/user/thingspeak && pwd && rails server webrick
```

Compile

```
chmod +x /home/linaro/runthingspeak.sh
```

And in **/etc/rc.local**,

```
su - user -c /home/user/runthingspeak.sh &
exit 0
```

or

```
sudo -u user sh /home/user/runthingspeak.sh &
```

3.5. Send TPH data to ThinkSpeak using GPRSbee

3.6. Additional Required Components

- GPRSbee Module
- MicroSim

3.7. Additional Required Libraries

- GPRSbee

Library Installation

The *GPRSbee* library is included with the SODAQ Mbili files that you have already installed.

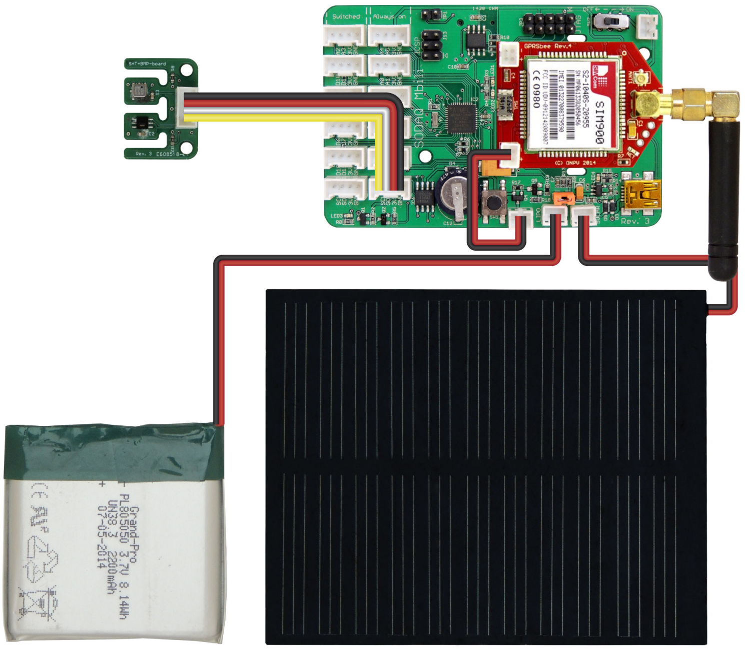
If necessary, refer to Section 2[7] of the Getting Started[8] guide for details on where to download from and how to install the SODAQ Mbili files.

3.8. Hardware Setup

You should refer to the board diagram[9], the Grove sockets page[10], and thehttp://mbili.sodaq.net/gprsbee-connection/[GPRSbee Connection] for additional information..

[7] http://mbili.sodaq.net/?page_id=23#step2
[8] http://mbili.sodaq.net/?page_id=23
[9] http://mbili.sodaq.net/?page_id=13

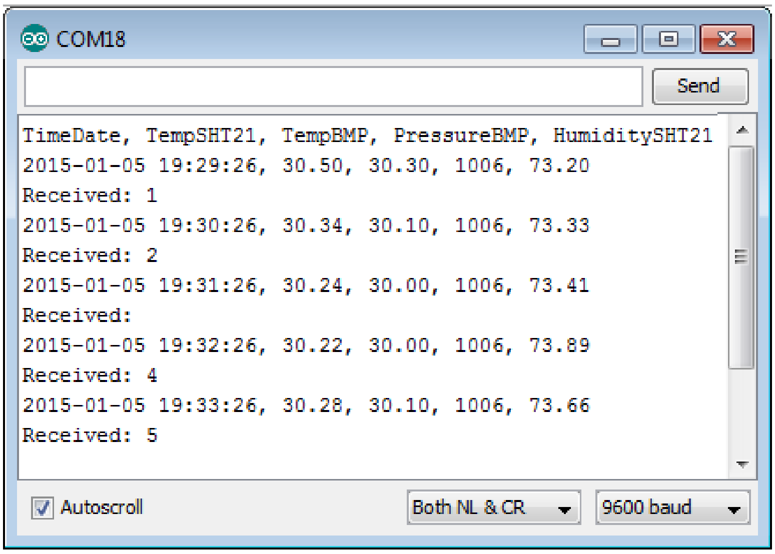
1. First, plug the TPH Sensor into the Grove I^2C socket.
2. Then, install the GPRSbee Module into the Bee socket.
3. Next, using the wiring diagram for the Switched Power Method[11], plug the 1A LiPo battery and GPRSbee power connectors into their sockets.
4. Finally, plug the 0.5W solar panel into its socket.



Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE onto the SODAQ Mbili board. Leave the USB cable plugged in and open the Serial Monitor (Ctrl-Shift-M) and ensure that it is set to the 9600 baud rate.

After you open the Serial Monitor (Ctrl-Shift-M), you should see output similar to this:

[10] http://mbili.sodaq.net/?page_id=81
[11] http://mbili.sodaq.net/gprsbee-connection/#switched



```
TimeDate, TempSHT21, TempBMP, PressureBMP, HumiditySHT21
2015-01-05 19:29:26, 30.50, 30.30, 1006, 73.20
Received: 1
2015-01-05 19:30:26, 30.34, 30.10, 1006, 73.33
Received: 2
2015-01-05 19:31:26, 30.24, 30.00, 1006, 73.41
Received:
2015-01-05 19:32:26, 30.22, 30.00, 1006, 73.89
Received: 4
2015-01-05 19:33:26, 30.28, 30.10, 1006, 73.66
Received: 5
```

3.9. Additional Sketch Code

Libarary Includes

In addition to the existing libraries, we must now also include the *GPRSbee* library in the sketch using the #include[12] compiler directive.

```
#include <GPRSbee.h>
```

Globals

The ThingSpeak[13] API limits data submission to a maximum of once every 15 seconds. Additionally, it takes some time to establish the GPRS connection before any data can be sent. For this reason, we adjust the *READ_DELAY* constant so that the readings are taken once per minute (the units are milliseconds).

We then define a series of constants which are used for setting up the GPRS connection and for sending the data to ThingSpeak[14]. The constants *APN, APN_USERNAME,* and *APN_PASSWORD* need to be set to the correct values for your particular network. Additionally, *WRITE_API_KEY*needs to be set to the Write API Key value shown on the *API*

[12] http://arduino.cc/en/Reference/Include
[13] https://thingspeak.com/
[14] https://thingspeak.com/

KEYS tab of your ThingSpeak[15] channel’s page. The other constants are used for formatting the data sent with the URL.

Note: The constants *LABELX* define the data labels for each of the fields. For the ThingSpeak[16] API you must use the labels *fieldN.* However, if you are modifying this example to work with another site you can change the labels here to suit your needs.

Additional note: You can also submit the data to "184.106.153.149/update”, this can be useful if your network is unable to resolve the specified URL. You should try this if you are receiving *603 DNS Error* responses (+HTTPACTION:0,603,0). You can see what the responses you are getting by enabling debugging (see the *setupComms()* section) and looking at the output in the Serial Monitor.

```
//The delay between the sensor readings
#define READ_DELAY 60000

//Network constants
#define APN "internet"
#define APN_USERNAME ""
#define APN_PASSWORD ""

//SpeakThings constants
#define URL "api.thingspeak.com/update"
#define WRITE_API_KEY "XXXXXXXXXXXXXXXX" //Change to your channel's key

//Seperators
#define FIRST_SEP "?"
#define OTHER_SEP "&"
#define LABEL_DATA_SEP "="

//Data labels, cannot change for ThingSpeak
#define LABEL1 "field1"
#define LABEL2 "field2"
#define LABEL3 "field3"
#define LABEL4 "field4"
```

setup()

In addition to the existing setup code, we make a call to the user defined method *setupComms()* which handles the initialisation of the GPRSbee Module.

[15] https://thingspeak.com/
[16] https://thingspeak.com/

```
//Setup GPRSbee
setupComms();
```

takeReading()

In addition to the existing code for taking the sensor readings, we also make a call to the user defined method *createDataURL()*. This method returns a String[17] containing the target URL as well as the sensor data in a format that can be sent directly with a HTTP request. We then send the URLhttp://arduino.cc/en/Reference/string[String] over the GPRS connection with a call to the user defined method *sendURLData()*.

```
//Get the data record as a URL
String url = createDataURL();

//Send it over the GPRS connection
sendURLData(url);
```

setupComms()

Here we initialise the GPRSbee Module. The Bee socket on the SODAQ Mbili is connected to the second serial port which is accessed through the *Serial1* object. We start with initialising *Serial1* with a call to Serial.begin()[18]. We then initialise the GPRSbee Module using the method*gprsbee.init().* The three parameters passed to this method include: the Serial object that the GPRSbee Module is connected to, the CTS pin (*BEECTS*), and the power pin (*BEEDTR*).

We must also make a call to *gprsbee.setPowerSwitchedOnOff()*, passing the argument *true*. This instructs the GPRSbee library to use the Switched Power Method[19]. (The method that we wired the GPRSbee and battery for in the *Hardware Setup* section.)

Note: If you need to debug the GPRSbee Module, you can uncomment the line which calls the method *gprsbee.setDiag()*. This method connects the output from *Serial1* to *Serial* so that any data sent over the *Serial1* connection is also sent over *Serial* and is displayed in the Serial Monitor.

```
void setupComms()
{
```

[17] http://arduino.cc/en/Reference/string
[18] http://arduino.cc/en/Serial/Begin
[19] http://mbili.sodaq.net/gprsbee-connection/#switched

```
  //Start Serial1 the Bee port
  Serial1.begin(9600);

  //Intialise the GPRSbee
  gprsbee.init(Serial1, BEECTS, BEEDTR);

  //uncomment this line to debug the GPRSbee with the serial monitor
  //gprsbee.setDiag(Serial);

  //This is required for the Switched Power method
  gprsbee.setPowerSwitchedOnOff(true);
}
```

3.10. createDataURL()

This method is similar in purpose to the existing user defined method *createDataRecord()*, which creates and returns a String[20] containing the sensor readings in CSV format. However, instead of the CSV format, this method constructs a String[21] which contains the target URL, the WRITE API KEY and the data from the sensor readings; all formatted for submission via a HTTP command to the ThingSpeak[22] website.

```
String createDataURL()
{
  //Construct data URL
  String url = URL;

  //Add key followed by each field
  url += String(FIRST_SEP) + String("key");
  url += String(LABEL_DATA_SEP) + String(WRITE_API_KEY);

  url += String(OTHER_SEP) + String(LABEL1);
  url += String(LABEL_DATA_SEP) + String(SHT2x.GetTemperature());

  url += String(OTHER_SEP) + String(LABEL2);
  url += String(LABEL_DATA_SEP) + String(bmp.readTemperature());

  url += String(OTHER_SEP) + String(LABEL3);
  url += String(LABEL_DATA_SEP) + String(bmp.readPressure() / 100);

  url += String(OTHER_SEP) + String(LABEL4);
  url += String(LABEL_DATA_SEP) + String(SHT2x.GetHumidity());
```

[20] http://arduino.cc/en/Reference/string
[21] http://arduino.cc/en/Reference/string
[22] https://thingspeak.com/

```
  return url;
}
```

sendURLData()

Here we send the constructed URL String[23] using a HTTP GET command. The reply from the ThingSpeak[24] website is normally the entry number of that submission. If the result is 0 the data was not successfully submitted. You may notice that sometimes nothing is received. This does not necessarily mean that the data was not successfully submitted, it just means that no answer was received before a timeout occurred.

```
void sendURLData(String url)
{
  char result[20] = "";
  gprsbee.doHTTPGET(APN, APN_USERNAME, APN_PASSWORD, url.c_str(), result,
 sizeof(result));

  Serial.println("Received: " + String(result));
}
```

3.11. Final Sketch Code

```
#include <Wire.h>
#include <SPI.h>
#include <SD.h>

//SODAQ Mbili libraries
#include <RTCTimer.h>
#include <Sodaq_BMP085.h>
#include <Sodaq_SHT2x.h>
#include <Sodaq_DS3231.h>
#include <GPRSbee.h>

//The delay between the sensor readings
#define READ_DELAY 60000

//Digital pin 11 is the MicroSD slave select pin on the Mbili
#define SD_SS_PIN 11

//The data log file
```

[23] http://arduino.cc/en/Reference/string
[24] https://thingspeak.com/

```
#define FILE_NAME "DataLog.txt"

//Data header
#define DATA_HEADER "TimeDate, TempSHT21, TempBMP, PressureBMP, HumiditySHT21"

//Network constants
#define APN "internet"
#define APN_USERNAME ""
#define APN_PASSWORD ""

//SpeakThings constants
#define URL "api.thingspeak.com/update"
#define WRITE_API_KEY "XXXXXXXXXXXXXXXX" //Change to your channel's key

//Seperators
#define FIRST_SEP "?"
#define OTHER_SEP "&"
#define LABEL_DATA_SEP "="

//Data labels, cannot change for ThingSpeak
#define LABEL1 "field1"
#define LABEL2 "field2"
#define LABEL3 "field3"
#define LABEL4 "field4"

//TPH BMP sensor
Sodaq_BMP085 bmp;

//RTC Timer
RTCTimer timer;

void setup()
{
  //Initialise the serial connection
  Serial.begin(9600);

  //Initialise sensors
  setupSensors();

  //Initialise log file
  setupLogFile();

  //Setup timer events
  setupTimer();

  //Setup GPRSbee
  setupComms();
  //Echo the data header to the serial connection
  Serial.println(DATA_HEADER);

  //Take first reading immediately
  takeReading(getNow());
}

void loop()
{
  //Update the timer
  timer.update();
}

void takeReading(uint32_t ts)
{
  //Create the data record
  String dataRec = createDataRecord();

  //Save the data record to the log file
  logData(dataRec);

  //Echo the data to the serial connection
  Serial.println(dataRec);

  //Get the data record as a URL
  String url = createDataURL();

  //Send it over the GPRS connection
  sendURLData(url);
}

void setupSensors()
{
  //Initialise the wire protocol for the TPH sensors
  Wire.begin();

  //Initialise the TPH BMP sensor
  bmp.begin();

  //Initialise the DS3231 RTC
  rtc.begin();
}

void setupLogFile()
{
  //Initialise the SD card
  if (!SD.begin(SD_SS_PIN))
  {
    Serial.println("Error: SD card failed to initialise or is missing.");
    //Hang
    while (true);
  }

  //Check if the file already exists
  bool oldFile = SD.exists(FILE_NAME);

  //Open the file in write mode
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Add header information if the file did not already exist
  if (!oldFile)
  {
    logFile.println(DATA_HEADER);
  }

  //Close the file to save it
  logFile.close();
}

void setupTimer()
{
  //Instruct the RTCTimer how to get the current time reading
  timer.setNowCallback(getNow);

  //Schedule the reading every second
  timer.every(READ_DELAY, takeReading);
}

void setupComms()
{
  //Start Serial1 the Bee port
  Serial1.begin(9600);

  //Intialise the GPRSbee
  gprsbee.init(Serial1, BEECTS, BEEDTR);

  //uncomment this line to debug the GPRSbee with the serial monitor
  //gprsbee.setDiag(Serial);

  //This is required for the Switched Power method
  gprsbee.setPowerSwitchedOnOff(true);
}
```

```
void logData(String rec)
{
  //Re-open the file
  File logFile = SD.open(FILE_NAME, FILE_WRITE);

  //Write the CSV data
  logFile.println(rec);

  //Close the file to save it
  logFile.close();
}

String createDataRecord()
{
  //Create a String type data record in csv format
  //TimeDate, TempSHT21, TempBMP, PressureBMP, HumiditySHT21
  String data = getDateTime() + ", ";
  data += String(SHT2x.GetTemperature())  + ", ";
  data += String(bmp.readTemperature()) + ", ";
  data += String(bmp.readPressure() / 100)  + ", ";
  data += String(SHT2x.GetHumidity());

  return data;
}

String createDataURL()
{
  //Construct data URL
  String url = URL;

  //Add key followed by each field
  url += String(FIRST_SEP) + String("key");
  url += String(LABEL_DATA_SEP) + String(WRITE_API_KEY);

  url += String(OTHER_SEP) + String(LABEL1);
  url += String(LABEL_DATA_SEP) + String(SHT2x.GetTemperature());

  url += String(OTHER_SEP) + String(LABEL2);
  url += String(LABEL_DATA_SEP) + String(bmp.readTemperature());

  url += String(OTHER_SEP) + String(LABEL3);
  url += String(LABEL_DATA_SEP) + String(bmp.readPressure() / 100);

  url += String(OTHER_SEP) + String(LABEL4);
  url += String(LABEL_DATA_SEP) + String(SHT2x.GetHumidity());

  return url;
}

void sendURLData(String url)
{
  char result[20] = "";
  gprsbee.doHTTPGET(APN, APN_USERNAME, APN_PASSWORD, url.c_str(), result,
 sizeof(result));

  Serial.println("Received: " + String(result));
}

String getDateTime()
{
  String dateTimeStr;

  //Create a DateTime object from the current time
  DateTime dt(rtc.makeDateTime(rtc.now().getEpoch()));

  //Convert it to a String
  dt.addToString(dateTimeStr);

  return dateTimeStr;
}

uint32_t getNow()
{
  return millis();
}
```

4. Low Power Mode - Sleep mode and RTC Interruption.

Mbili boards uses 40mA during normal operation. A rechargable Li-ion battery will have a typical capacity of 1000 mAh. If used to power Mbili board, it will last for about **25 hours** (100mAh/40mA) under normal operation

For a longer battery operation, another options must be taking into account:

- Use a battery with a bigger capacity, or use a battery pack, made up of several batteries.
- Use a solar panel with a battery pack to be charged during daylight.
- Put microcontroller into sleep mode cycling the operation.

In our case, Mbili will be sensing every period of time, ex. every 10 minutes. So an smart solution would be to put the microcontroller into sleep mode for the rest of the time while it is doing nothing.

As Mbili has an RTC onboard we use a RTC interruption to wake it up.

Also note that the power consumption will be influenced by any external circuit that is connected to Mbili board. So a good practice would be also to switch sensors and communication devices off during sleeping periods

4.1. Sleep mode

The ATmega1284 micro-controller in our Mbili board supports several modes of sleep:

- SLEEP_MODE_IDLE - the least power savings
- SLEEP_MODE_ADC
- SLEEP_MODE_PWR_SAVE
- SLEEP_MODE_STANDBY
- SLEEP_MODE_PWR_DOWN - the most power savings

The more power saving the sleep mode provides, the less functionality is active.

E.g. in **Power-Down** sleep mode, only the external interrupt and watch dog timer (WDT) are active, in **Idle** sleep mode the UART, timers, ADC, etc are all active, just the CPU and Flash clocks are disabled. See Section 10 of the ATmega1284 datasheet[25] for more information.

Table 10-1 shows the different sleep modes, their wake up sources and Brown-out Detector (BOD) disable ability

When enabled, the BOD actively monitors the power supply voltage during the sleep periods. To further save power, it is possible to disable the BOD in some sleep modes.

4.2. Header files and general information

To use the watchdog timer, a sketch needs to include three header files:

```
#include <avr/sleep.h>
#include <avr/power.h>
```

[25] https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&sqi=2&ved=0CB0QFjAA&url=http%3A%2F%2Fwww.atmel.com%2Fimages%2Fdoc8059.pdf&ei=14JQVZSwHoHjUvWagJAC&usg=AFQjCNGhbgfnfU0c3f6zyQxo6C2PWY797Q&sig2=B-kpP6RFzxbUievQ8xCAOA&bvm=bv.92885102,d.bGQ

These provide definitions for various functions and variables needed to control the watchdog timer and manage some of the other power functions.

There are several Arduino library functions used to control sleep mode. They are:

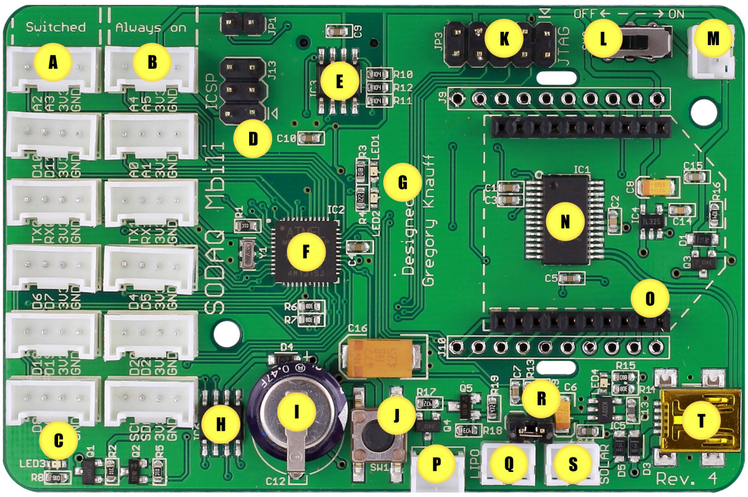
- set_sleep_mode(mode) - Configures the Atmega168 for the specified sleep mode (see above for supported sleep modes);
- sleep_enable() - Enables the sleep mode to be entered;
- sleep_mode() - Enters the sleep mode. Before this is called, the appropriate mechanism for waking the microcontroller must have been set up;
- sleep_disable() - Disables the sleep mode;

4.3. Test minimum current consumption

4.4. Sketch Code

```
#include <avr/power.h>
#include <avr/sleep.h>

void setup() {
  enterSleep();
}

void loop() {
}

void enterSleep(void)
{
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();

  ADCSRA &= ~(1<<ADEN);              //Disable ADC
  ACSR = (1<<ACD);                   //Disable the analog comparator
  DIDR0 = 0x3F;                      //Disable digital input buffers on all ADC0-ADC5
 pins
  DIDR1 = (1<<AIN1D)|(1<<AIN0D);    //Disable digital input buffer on AIN1/0

  power_twi_disable();
  power_spi_disable();
  power_usart0_disable();
  power_timer0_disable();     //Needed for delay_ms
  power_timer1_disable();

  sleep_mode();
  /** The program will continue from here. **/
  /* First thing to do is disable sleep. */
  sleep_disable();
}
```

Using this code, connecting the battery and taking Battery power draw measurement jumper out (jumper R). It is possible to measure consumption.



Testing with a multimeter on the jumper

Consumption when microcontroller is in sleeping mode is 220uA.

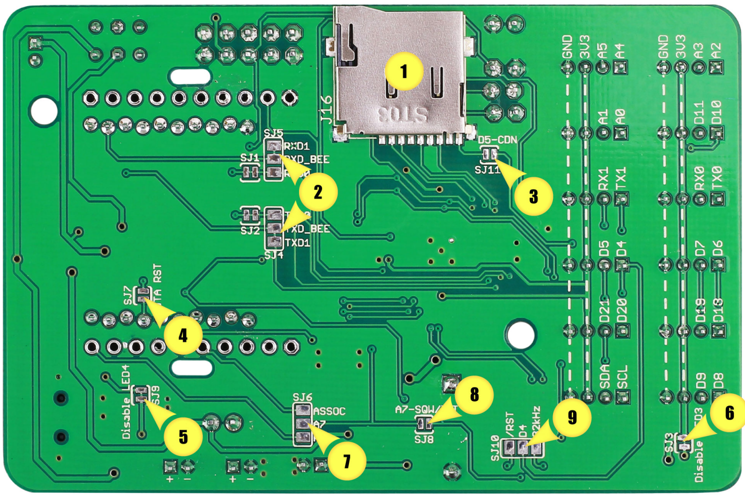
So the maximum operation period is more than 6 months of operation. This is just a to know which is the minimum consumption, remember power consumption will be influenced by microcontroller operation and any external circuit that is connected to Mbili board.

4.5. RTC interruption

It is not very usefull to have the board alll the time in sleeping mode. We want to wake it up in a period of time measure, switch on sensors, log data and send it using a communication board (Xbee, Wifi, GPRS, satellite, etc.). Once operation is done. switch everythig off and put microcontroller in sleep mode again and wait untill next measurement period.

Waking the microcontroller up is possible using RTC interruptions as follow

To enable RTC interruption Mbili jumper 8 must be solder. It allows the RTC to be used as an interrupt device.



So an example of operation is shown in the following plot. Peaks can be seen every time microcontroller wakes up.

4.6. TPH measurement in low power mode

```
#include <Wire.h>
#include <avr/sleep.h>
#include <avr/wdt.h>

//SODAQ Mbili libraries
#include <RTCTimer.h>
#include <Sodaq_BMP085.h>
#include <Sodaq_SHT2x.h>
#include <Sodaq_DS3231.h>
#include <GPRSbee.h>
#include <Sodaq_PcInt.h>

//The sleep length in seconds (MAX 86399)
#define SLEEP_PERIOD 10

//RTC Interrupt pin and period
#define RTC_PIN A7

//Data header
#define DATA_HEADER "TimeDate, TempSHT21, TempBMP, PressureBMP, HumiditySHT21"
```

```
//TPH BMP sensor
Sodaq_BMP085 bmp;

void setup()
{
  //Initialise the serial connection
  Serial.begin(9600);
  Serial.println("Power up...");

  //Initialise sensors
  setupSensors();

  //Setup sleep mode
  setupSleep();

  //Echo the data header to the serial connection
  Serial.println(DATA_HEADER);
}

void loop()
{
  //take readings
  takeReading();

  //Sleep
  systemSleep();
}

void setupSleep()
{
  pinMode(RTC_PIN, INPUT_PULLUP);
  PcInt::attachInterrupt(RTC_PIN, wakeISR);

  //Setup the RTC in interrupt mode
  rtc.begin();

  //Set the sleep mode
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
}

void wakeISR()
{
  //Leave this blank
}

void systemSleep()
```

```
{
  Serial.print("Sleeping mode for ");
  Serial.print(SLEEP_PERIOD/60.0);
  Serial.println(" minutes");

  //Wait until the serial ports have finished transmitting
  Serial.flush();
  Serial1.flush();

  //Schedule the next wake up pulse timeStamp + SLEEP_PERIOD
  DateTime wakeTime(getNow() + SLEEP_PERIOD);
  rtc.enableInterrupts(wakeTime.hour(), wakeTime.minute(), wakeTime.second());

  //The next timed interrupt will not be sent until this is cleared
  rtc.clearINTStatus();

  //Disable ADC
  ADCSRA &= ~_BV(ADEN);

  //Sleep time
  noInterrupts();
  sleep_enable();
  interrupts();
  sleep_cpu();
  sleep_disable();

  //Enbale ADC
  ADCSRA |= _BV(ADEN);
  Serial.println("Waking-up");
  //This method handles any sensor specific wake setup
}

//void takeReading(uint32_t ts)
void takeReading()
{
  //Create the data record
  String dataRec = createDataRecord();

  //Echo the data to the serial connection
  Serial.println(dataRec);
}

void setupSensors()
{
  //Initialise the wire protocol for the TPH sensors
  Wire.begin();
  //Initialise the TPH BMP sensor
  bmp.begin();

  //Initialise the DS3231 RTC
  rtc.begin();
}

String createDataRecord()
{
  //Create a String type data record in csv format
  //TimeDate, TempSHT21, TempBMP, PressureBMP, HumiditySHT21
  String data = getDateTime() + ", ";
  data += String(SHT2x.GetTemperature())  + ", ";
  data += String(bmp.readTemperature()) + ", ";
  data += String(bmp.readPressure() / 100)  + ", ";
  data += String(SHT2x.GetHumidity());

  return data;
}

String getDateTime()
{
  String dateTimeStr;

  //Create a DateTime object from the current time
  DateTime dt(rtc.makeDateTime(rtc.now().getEpoch()));

  //Convert it to a String
  dt.addToString(dateTimeStr);

  return dateTimeStr;
}

uint32_t getNow()
{
  return rtc.now().getEpoch();
}
```