Laboratories of Day 2

Table 1. Day two

Laboratory	Code
1: Sending an SMS message with GPRSbee	Send_SMS.ino
2: Sending an SMS with Date, Time, RTC Temperature and Battery Voltage	Send_SMS_DateTemp.ino
3: Syncing the RTC	RTC_update.ino
4: Configuring Zigbee radios	No code for this lab
5: Sending data between two devices using Zigbee	RTC_date_Volt_Temp_Zigbee.ino
6: Connecting an Analog Sensor	SensorAD0.ino, Pote_angle.ino
7: Connecting a Digital Sensor	Switch_relay.ino, Digital_TPH.ino
8: Using a Button to Activate a Buzzer	Lab2.9.ino
9: Turning ON a Light When It Gets Dark	Lab2.10.ino
10: Using an Ultrasonic Ranger to Measure Distance	Lab2.11.ino
11: Using Temperature, Humidity & Moisture Sensor	Hum_temp_moist.ino

1. Sending an SMS message with GPRSbee

The GPRSbee is a GPRS/GSM expansion board, designed by Gregory Knauff

It is an alternative for the Arduino GPRS Shield and has the Xbee form factor so it can be used in any system that has a bee socket like the Seeeduino Stalker, the Arduino Fio and the SODAQ. The GPRSbee uses SIM cards of the MicroSIM form factor.

The core of this board is the well known SIM900 module. This module, like most other GPRS/GSM modules, has an operating voltage of 3.5 - 4.5 volt and can draw up to 2A power during broadcasts bursts. This makes the 3.3V power that the bee socket can provide unsuitable. This has been solved by powering the GPRSbee directly from a 3.7 volt LiPo battery.

To accomodate this the GPRSbee has two JST sockets, one to connect the battery and a second one to provide the power to the main board. When the main board has a LiPo charge circuit (like the Stalker and the Fio have), this allows for charging the LiPo battery too.

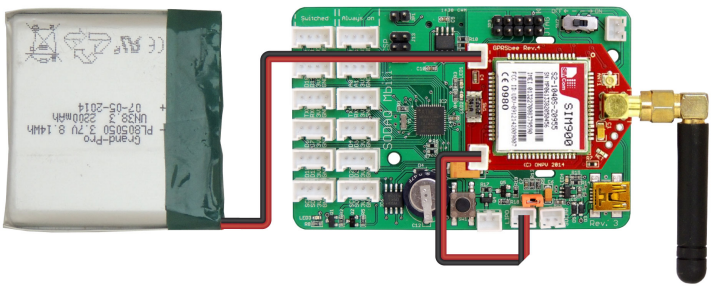
1.1. GPRSbee Connection

As the module draws a significant amount of power, we want to switch it off when unused. There are two methods for controlling the on/off state of the GPRSbee. The “old” method is to toggle DTR which toggles the on-off state of the SIM900. The “new” method is to switch the power supply of GPRSbee. This new method is only supported by SODAQ Mbili. The SODAQ Mbili board has a connector (JP2) which is switched on or off by setting *GPRSbeePower* (D23) to HIGH or LOW.

The On-Off Toggle Method

The power (battery) connections are as follows:

- Connect the battery to one of the GPRSbee power connectors.
- Connect the other GPRSbee power connector to the LiPo connector on the SODAQ board.



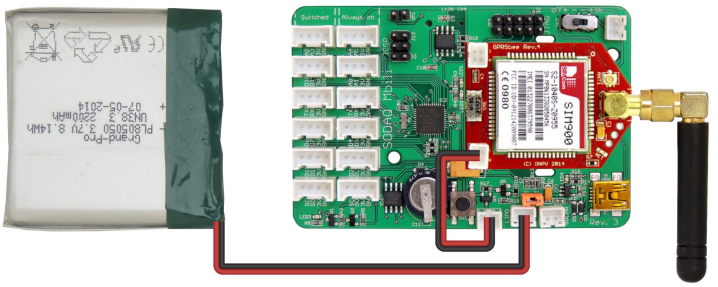
The default setting for the GPRSbee library is to use this mode.

The Switched Power Method

This method is only supported by the SODAQ Mbili.

The power (battery) connections are as follows:

- Connect the battery to the LiPo connector on the SODAQ Mbili board.
- Connect the SODAQ Mbili JP2 connector to one of the power connectors on the GPRSbee.



To enable this mode you must add the following to your setup or initialisation code:

```
gprsbee.setPowerSwitchedOnOff(true);
```

Note: Calling the above method and passing the parameter *false*, will switch the mode back to the *On-Off Toggle* method.

1.2. What is GSM

GSM is an international standard for mobile telephones. It is an acronym that stands for Global System for Mobile Communications. It is also sometimes referred to as 2G, as it is a second-generation cellular network.

To use GPRS for internet access, and for the Arduino to request or serve webpages, you need to obtain the Access Point Name (APN) and a username/password from the network operator. See the information in Connecting to the Internet for more information about using the data capabilities of the shield.

Among other things, GSM supports outgoing and incoming voice calls, Simple Message System (SMS or text messaging), and data communication (via GPRS).

The GPRSbee module is a a GSM modem. From the mobile operator perspective, it looks just like a mobile phone. From the Arduino perspective, it looks just like a modem.

What is GPRS

GPRS is a packet switching technology that stands for General Packet Radio Service. It can provide idealized data rates between 56-114 kbit per second. With the GPRSbee module, it is possible to leverage the data communication to access the Internet. Similar to the Ethernet and WiFi libraries, the GSM library allows the Arduino to act as a client or server, using http calls to send and receive web pages.

Network operator requirements

To access a network, you must have a subscription with a mobile phone operator (either prepaid or contract), a GSM compliant device like the GPRSbee, and a Subscriber Identity Module (SIM) card. The network operator provides the SIM card, which contains information like the mobile number, and can store limited amounts of contacts and SMS messages.

It's common for SIM cards to have a four-digit PIN number associated with them for security purposes. Keep note of this number, as it's necessary for connecting to a network. If you lose the PIN associated with your SIM card, you may need to contact your network operator to retrieve it. *Some SIM cards become locked if an incorrect PIN is entered too many times. If you're unsure of what the PIN is, look at the documentation that came with your SIM.* Using a PUK (PIN Unlock Code), it is possible to reset a lost PIN with the GSM shield and an Arduino. The PUK number will come with your SIM card documentation.

There are a few different sizes of SIM cards; the GPRSbee accepts cards in the micro-SIM format.

To use GPRS for internet access, and for the Arduino to request or serve webpages, you need to obtain the Access Point Name (APN) and a username/password from the network operator.

1.3. Sending an SMS

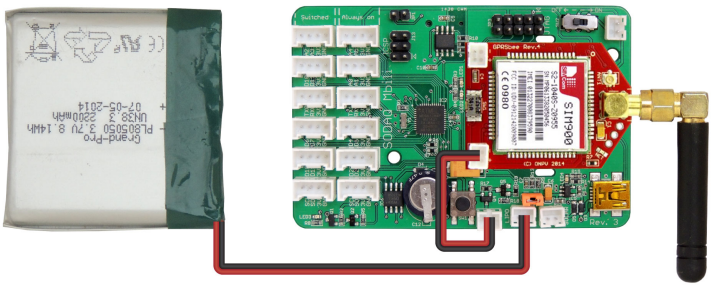
In this example we will connect the GPRSBee board to the SODAQ Mbili and send an SMS text with the message "Hello World".

The GPRSBee board uses the GPRSbee library to send SMS text messages and to connect via 2G or 3G. The *GPRSbee* librarie is included with the SODAQ Mbili files that you have already installed.

Table 2. SMS example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack, GPRSBee board and Micro SIM card
Required Libraries	GPRSbee
Hardware Setup	Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets. Insert the GPRSbee board after inserting the Micro SIM card.
Source Code	**Send_SMS.ino**

Insert the Micro Sim card in the GPRSBee board and connect as follows.



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.

Send_SMS.ino

```
//SODAQ Mbili libraries
#include <GPRSbee.h>
```

```
// Fill in your mobile number here. Start with + and country code
#define TELNO      "+393898896252"

void setup ()
{
     Serial.begin(9600);      // Serial1 is connected to SIM900 GPRSbee
     Serial.println("sending a SMS text: Hello world");

     //Setup GPRSbee
     setupComms();
}

void loop ()
{
    bool smsSent = gprsbee.sendSMS(TELNO,"Hello World");
    delay(10000);
}


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);
}
```

- Here the necessary library files are included in the sketch using the #include[1] compiler directive.

```
//SODAQ Mbili libraries
#include <GPRSbee.h>
```

- Here we define the telephone number. Remember to add + and country code.

[1] http://arduino.cc/en/Reference/Include

```
// Fill in your mobile number here. Start with + and country code
#define TELNO      "+393898896252"
```

- Here we start by initialising the serial connection with a call to Serial.begin()[2] and print the message "sending a SMS text: Hello world". Then we call setupComms function.

```
void setup ()
{
     Serial.begin(9600);      // Serial1 is connected to SIM900 GPRSbee
     Serial.println("sending a SMS text: Hello world");

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

- Here we use the GPRSbee library that sends SMS with the sentence "Hello world"

```
void loop ()
{
   bool smsSent = gprsbee.sendSMS(TELNO,"Hello world");
   delay(10000);
}
```

- This function initializes Serial port number 1, then initializes GPRSBee and configures Switched Power method. Remember here you can uncomment *gprsbee.setDiag(Serial);* to debug the GPRSbee with the serial monitor

```
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
```

[2] http://arduino.cc/en/Serial/Begin

```
    gprsbee.setPowerSwitchedOnOff(true);
}
```

2. Sending an SMS with Date, Time, RTC Temperature and Battery Voltage

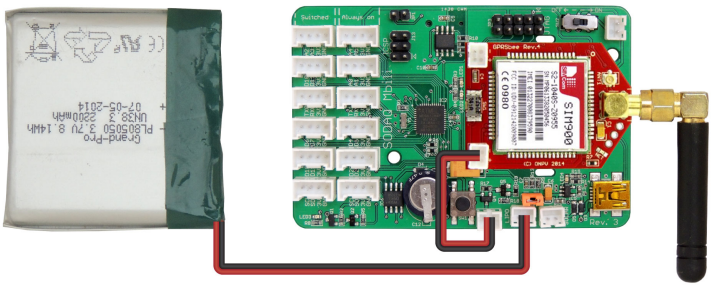
In this example we will connect GPRSBee board to Mbili and send an SMS text with a message which contains Date, Time, RTC Temperature and Battery Voltage.

Table 3. SMS with Date, Time, Temperature and Battery example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack, GPRSBee board and Micro SIM card
Required Libraries	GPRSbee
Hardware Setup	Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets. Insert the GPRSbee board after inserting the Micro SIM card.
Source Code	**Send_SMS_DateTemp.ino**

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

Put the Micro SIM card in the GPRSBee board and connect as follows.



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.

Send_SMS_DateTemp.ino

```
#include <Wire.h>

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

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

//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

// Fill in your mobile number here. Start with + and country code
#define TELNO      "+393898896252"

Sodaq_DS3231 RTC; //Create RTC object for DS3231 RTC come Temp Sensor
//year, month, date, hour, min, sec and week-day(starts from 0 and goes to 6)
//writing any non-existent time-data may interfere with normal operation of the RTC.
//Take care of week-day also.*/
DateTime dt(2015, 06, 05, 11, 5, 00, 4);

void setup ()
{
     //Initialise the serial connection
     Serial.begin(9600);      // Serial1 is connected to SIM900 GPRSbee
     Serial.println("Starting");

     //Initialise sensors
     setupSensors();

     //Setup GPRSbee
     setupComms();
}

void loop ()
{
    //Create the data
    String dataRec = createData();
    Serial.println("Sending SMS with value: "+ dataRec);
    bool smsSent = gprsbee.sendSMS(TELNO,dataRec.c_str()); //String.c_str() send
 Char array of String
```

```
    delay(SEND_DELAY);
}

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

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

  //remember to comment this line once RTC is updated
  rtc.setDateTime(dt); //Adjust date-time as defined 'dt' above
}

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);
}

String createData()
{
  //Create a String type data record in csv format
  ///Read the temperature
  RTC.convertTemperature();
  float temp = RTC.getTemperature();

  // Convert temperature voltage to string
  char buffer[14]; //make buffer large enough for 7 digits
  String temperatureS = dtostrf(temp, 7,2,buffer);
  //'7' digits including '-' negative, decimal and white space. '2' decimal places
  temperatureS.trim(); //trim whitespace, important so Ubidots will treat it as a
 number

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

```
  String data = getDateTime()+ ", Temp=";
  data += String(temperatureS)+ "C, Volt=";
  data += String(mv)+ "mV";

  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;
}

float getRealBatteryVoltage()
{
  uint16_t batteryVoltage = analogRead(BATVOLTPIN);
  return (ADC_AREF / 1023.0) * (BATVOLT_R1 + BATVOLT_R2) / BATVOLT_R2 *
 batteryVoltage;
}
```

- Here the necessary library files are included in the sketch using the #include[3] compiler directive.

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

//SODAQ Mbili libraries
#include <Sodaq_DS3231.h>
#include <GPRSbee.h>
```

- Here we define the telephone number. Remember to add + and country code. We also define constants are used for reading the battery voltage and delay between messages sent. We create RTC object for DS3231 RTC come Temperature Sensor.

[3] http://arduino.cc/en/Reference/Include

```
// Fill in your mobile number here. Start with + and country code
#define TELNO      "+393898896252"

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

//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

Sodaq_DS3231 RTC; //Create RTC object for DS3231 RTC come Temp Sensor
//year, month, date, hour, min, sec and week-day(starts from 0 and goes to 6)
//writing any non-existent time-data may interfere with normal operation of the RTC.
//Take care of week-day also.*/
DateTime dt(2015, 06, 05, 11, 5, 00, 4);
```

- The difference with the previous example is that here we create an String *dataRec* with the data values. Then we use GPRSBee library *gprsbee.sendSMS* to send an SMS text.

Note: We have to add *dataRec.c_str()* that is the char values from the String object. *dataRec.c_str()* is representing the current value of the string[4] object.

```
void loop ()
{
    //Create the data
    String dataRec = createData();
    Serial.println("Sending SMS with value: "+ dataRec);
    bool smsSent = gprsbee.sendSMS(TELNO,dataRec.c_str()); //String.c_str() send
 Char array of String
    delay(SEND_DELAY);
}
```

- This function takes readings from the battery voltage, Date and Time, and temperature from RTC. With this values creates an String *data* to be sent as an SMS text.

```
String createData()
{
  //Create a String type data record in csv format
  ///Read the temperature
```

[4] http://www.cplusplus.com/string

```
  RTC.convertTemperature();
  float temp = RTC.getTemperature();

  // Convert temperature voltage to string
  char buffer[14]; //make buffer large enough for 7 digits
  String temperatureS = dtostrf(temp, 7,2,buffer);
  //'7' digits including '-' negative, decimal and white space. '2' decimal places
  temperatureS.trim(); //trim whitespace, important so Ubidots will treat it as a
 number

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

  String data = getDateTime()+ ", Temp=";
  data += String(temperatureS)+ "C, Volt=";
  data += String(mv)+ "mV";

  return data;
}
```

3. Syncing the RTC

In this example we will be using the GPRSbee module to connect to the SODAQ time server[5] in order to retrieve the current date and time stamp for the purpose of updating the internal Real Time Clock (RTC) on the SODAQ Mbili board.

If you navigate your browser to the SODAQ time server at: http://time.sodaq.net/ you will see a numeric value shown in the upper left hand corner of the screen. This value represents the current UTC time, specified in seconds since the start of Epoch time[6] (00:00:00 01/01/1970). Here this value is retrieved and used to update the DS3231 RTC time stamp.

Table 4. Syncing the RTC example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack, GPRSBee board and Micro SIM card
Required Libraries	GPRSbee, Wire, Sodaq_DS3231
Hardware Setup	Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets. Insert
Source Code	**RTC_update.ino**

[5] http://time.sodaq.net/
[6] http://en.wikipedia.org/wiki/Unix_time

	the GPRSbee board after inserting the Micro SIM card.
Source Code	**RTC_update.ino**

The *Wire* Library comes pre-installed with the Arduino IDE, and so there is no need to download or install it. The *Sodaq_DS3231* and the *GPRSbee* libraries are included with the SODAQ Mbili files that you have already installed.

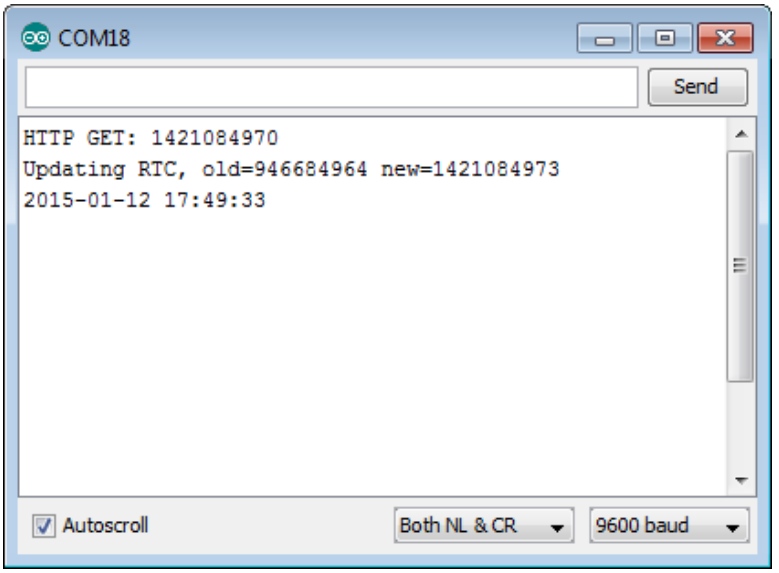
If necessary, refer to the Getting Started guide for details on where to download from and how to install the SODAQ Mbili files.

Install the GPRSbee module into the Xbee socket. Following the wiring diagram for the Switched Power Method [7], plug the 1A LiPo battery and GPRSbee power connectors into their sockets. Finally, plug the 0.5W solar panel into its socket as shown below.

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:

[7] http://mbili.sodaq.net/gprsbee-connection/#switched



If the current time stamp of the RTC is within about 30 seconds of the retrieved time stamp, the RTC will not be updated and you will not see the second line of output.

RTC_update.ino

```
#include <Wire.h>
#include <Sodaq_DS3231.h>
#include <GPRSbee.h>

#define APN "internet"
#define APN_USERNAME ""
#define APN_PASSWORD ""

#define TIME_URL "time.sodaq.net"
#define TIME_ZONE 0.0
#define TIME_ZONE_SEC (TIME_ZONE * 3600)

void setup()
{
  //Start Serial for serial monitor
  Serial.begin(9600);

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

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

```
  //Uncomment the line below to debug the GPRSbee with the serial monitor
  //gprsbee.setDiag(Serial);

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

  //Sync time
  syncRTCwithServer();

  //Print out new date/time
  Serial.println(getDateTime());
}

void loop()
{
}

void syncRTCwithServer()
{
  char buffer[20];

  if (gprsbee.doHTTPGET(APN, APN_USERNAME, APN_PASSWORD, TIME_URL, buffer,
 sizeof(buffer)))
  {
    Serial.println("HTTP GET: " + String(buffer));

    //Convert the time stamp to unsigned long
    char *ptr;
    uint32_t newTs = strtoul(buffer, &ptr, 0);

    //Add the timezone difference plus a few seconds
    //to compensate for transmission and processing delay
    newTs += 3 + TIME_ZONE_SEC;

    //If conversion was successful
    if (ptr != buffer)
    {
      //Get the old time stamp
      uint32_t oldTs = rtc.now().getEpoch();
      int32_t diffTs = abs(newTs - oldTs);

      //If time is more than 30s off, update
      if (diffTs > 30)
      {
        //Display old and new time stamps
        Serial.print("Updating RTC, old=" + String(oldTs));
```

```
        Serial.println(" new=" + String(newTs));

        //Update the rtc
        rtc.setEpoch(newTs);
      }
    }
  }
}

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;
}
```

- Here the necessary library files are included in the sketch using the #include[8] compiler directive.

```
#include <Wire.h>
#include <Sodaq_DS3231.h>
#include <GPRSbee.h>
```

- Here we specify the Access Point Name (APN), APN Username, and APN Password for the network that the GPRSbee will connect to. You will need to change these APN values to those required by your specific network. Next we specify the URL for the time server which will return the current UTC time stamp. You can modify *TIME_ZONE* to reflect your current time zone, specified in hours (or part hours) +/- relative to UTC.

```
#define APN "internet"
#define APN_USERNAME ""
#define APN_PASSWORD ""

#define TIME_URL "time.sodaq.net"
#define TIME_ZONE 0.0
```

[8] http://arduino.cc/en/Reference/Include

```
#define TIME_ZONE_SEC (TIME_ZONE * 3600)
```

- On the SODAQ Mbili board, the serial connection to the PC is connected to the first serial port which is accessed through the *Serial* object, and the Bee socket is connected to the second serial port which is accessed through the *Serial1* object.

We start with initialising both *Serial* and *Serial1* with calls to Serial.begin()[9]. 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[10]. (The method that we wired the GPRSbee and battery for in the *Hardware Setup* section.)

Next we call the user defined method *syncRTCwithServer()* which retrieves the current time stamp from the server and updates the RTC. Finally, we print the updated date and time using a reading from the RTC.

```
void setup()
{
  //Start Serial for serial monitor
  Serial.begin(9600);

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

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

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

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

  //Sync time
  syncRTCwithServer();

  //Print out new date/time
```

[9] http://arduino.cc/en/Serial/Begin
[10] http://mbili.sodaq.net/gprsbee-connection/#switched

```
  Serial.println(getDateTime());
}
```

- The code in this sketch attempts to synchronise the RTC once when the *setup()* method is called. No further code is executed and so the *loop()* method is emptly.

```
void loop()
{
}
```

- Here we attempt a HTTP GET request with the *gprsbee.doHTTPGET()* passing the parameters for the APN, APN_USERNAME, APN_PASSWORD, the URL, a return buffer and the size of that return buffer. If the HTTP GET request was successful, we then output the returned data (stored in *buffer*) to the Serial Monitor.

The returned data is an ASCII time stamp specifying the number of seconds in Epoch time[11]. We then convert the value to an unsigned long integer (using strtoul()[12]) and add both the timezone seconds as well as a few additional seconds to compensate for any transmission and processing delays.

If the conversion was successful, we then check if the old time stamp from the RTC is more than 30 seconds different from the new time stamp from the time server. If so, we then display both the old and new time stamps in the Serial Monitor and then update the RTC with the new time stamp using the method *rtc.setEpoch()*.

```
void syncRTCwithServer()
{
  char buffer[20];

  if (gprsbee.doHTTPGET(APN, APN_USERNAME, APN_PASSWORD, TIME_URL, buffer,
 sizeof(buffer)))
  {
    Serial.println("HTTP GET: " + String(buffer));

    //Convert the time stamp to unsigned long
    char *ptr;
    uint32_t newTs = strtoul(buffer, &ptr, 0);
```

[11] http://en.wikipedia.org/wiki/Unix_time
[12] http://www.cplusplus.com/reference/cstdlib/strtoul/

```
    //Add the timezone difference plus a few seconds
    //to compensate for transmission and processing delay
    newTs += 3 + TIME_ZONE_SEC;

    //If conversion was successful
    if (ptr != buffer)
    {
      //Get the old time stamp
      uint32_t oldTs = rtc.now().getEpoch();
      int32_t diffTs = abs(newTs - oldTs);

      //If time is more than 30s off, update
      if (diffTs > 30)
      {
        //Display old and new time stamps
        Serial.print("Updating RTC, old=" + String(oldTs));
        Serial.println(" new=" + String(newTs));

        //Update the rtc
        rtc.setEpoch(newTs);
      }
    }
  }
}
```

- Here we return a Date & Time reading from the RTC in a String[13] format. First a DateTime[14] object is constructed using a time reading from the DS3231 RTC. This is then converted into a String[15] using the method*DateTime.addToString()* the result of which is then returned from this method.

```
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;
```

[13] http://arduino.cc/en/Reference/string
[14] http://playground.arduino.cc/Code/DateTime
[15] http://arduino.cc/en/Reference/string

```
}
```

4. Configuring Zigbee radios

4.1. X-CTU

X-CTU[16] was developed by Digi and it is only available for Windows. This is the version 2 of the tutorial, modified to fit the new 2014 X-CTU version.

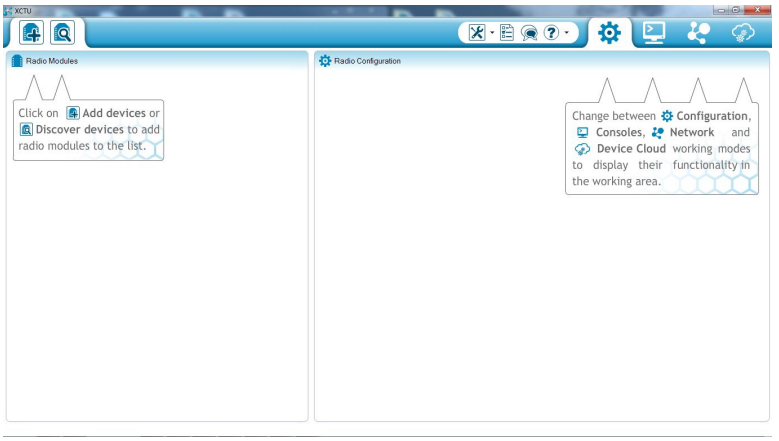
Once X-CTU has been downloaded, the next step is to install the program. When the program asks for updating from Digi, we must answer ‘yes’ so as to download all the firmware versions for all the XBee modules.

> Changing or upgrading the XBee firmware is a delicate process that may harm permanently the XBee module.

When X-CTU has been properly installed, the Zigbee module can be connected to the computer using the Xbee adapter.

It will be recognized as a ‘USB Serial Port’. We have to know the COM number given to this device in order to specify it in the X-CTU (in our test, COM1 was the value given by Windows, as seen later in Figure 2).

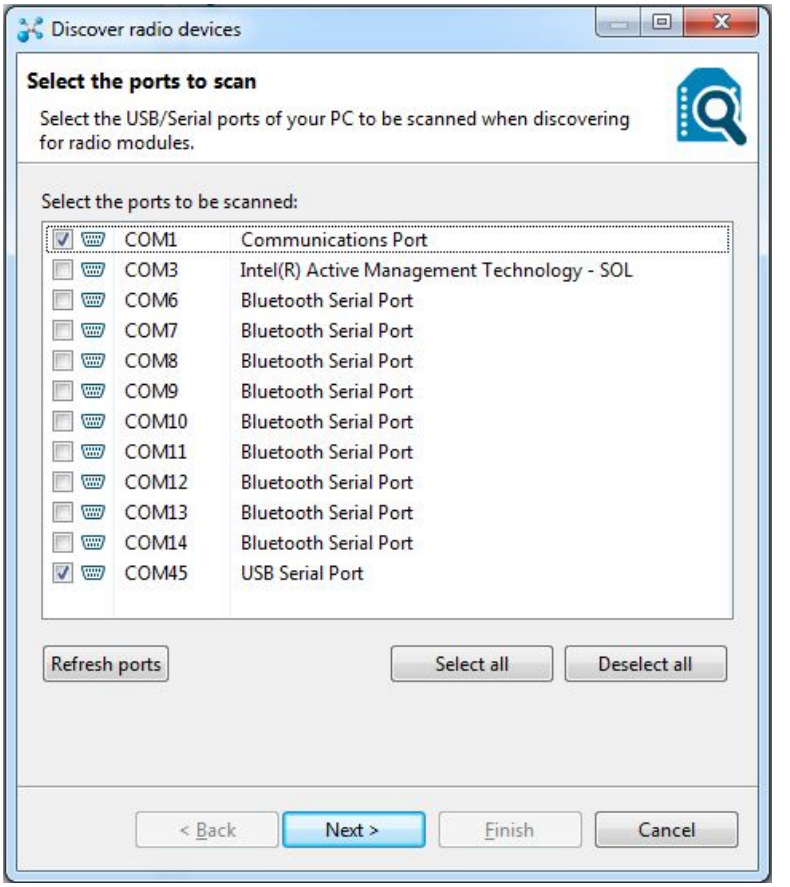
Finally, we launch X-CTU and the program will start. A window like the one below will appear, showing the different functions and the different COM ports detected.

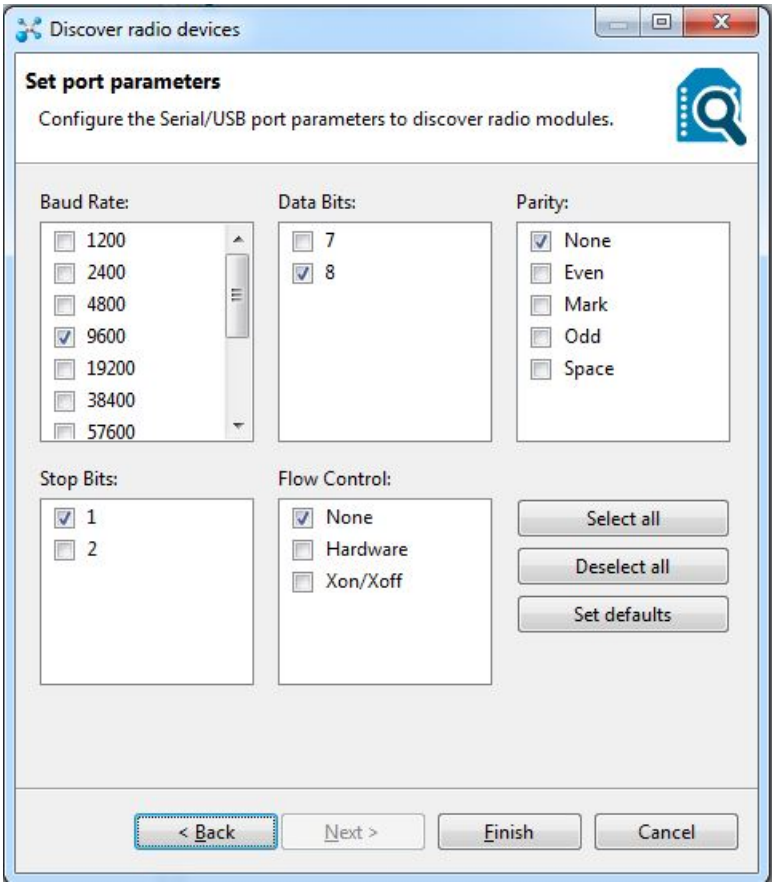


[16] http://www.digi.com/support/productdetl.jsp?pid=3352&osvid=57&tp=5&s=316

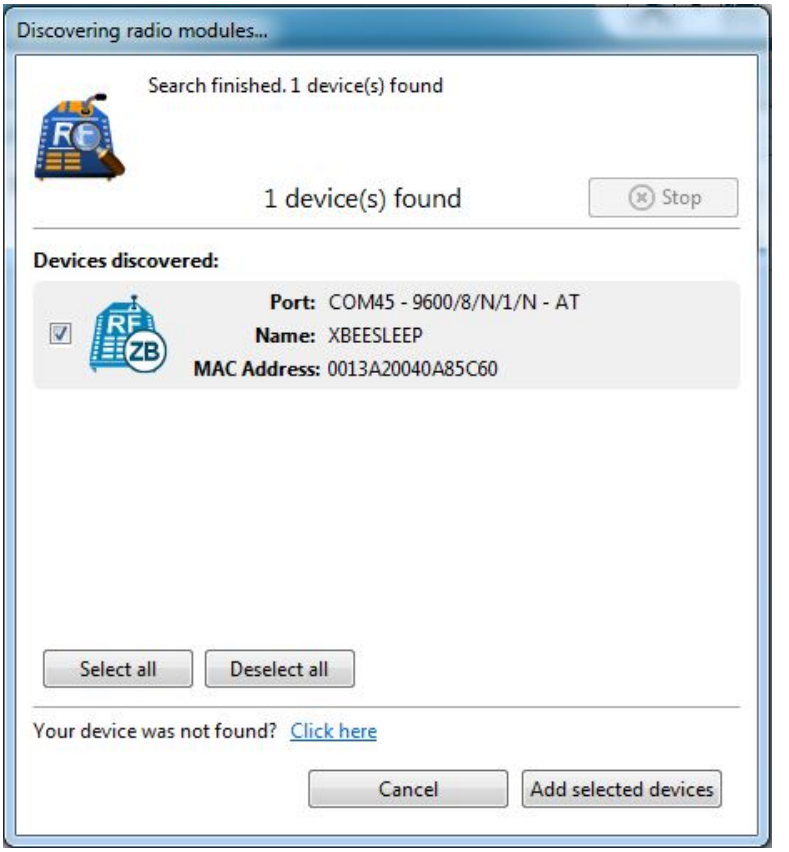
4.2. X-CTU operations

Press the button “Discover radio modules connected” (on the top left, with a magnifying glass), select the appropriate communication port and configure it as shown below (if you are not sure, you can select all).

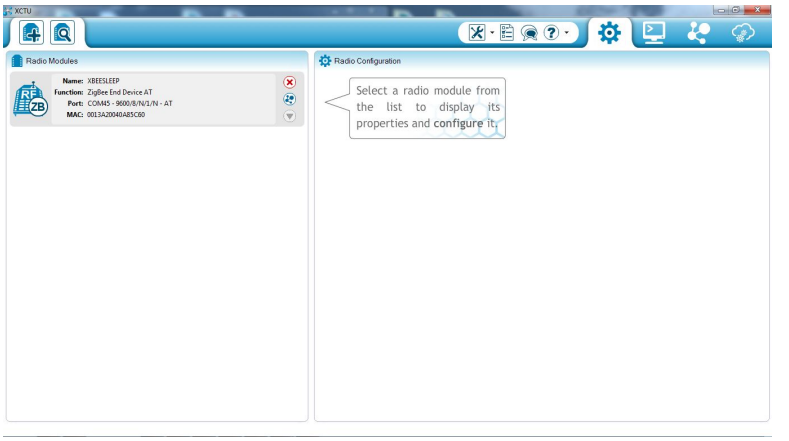




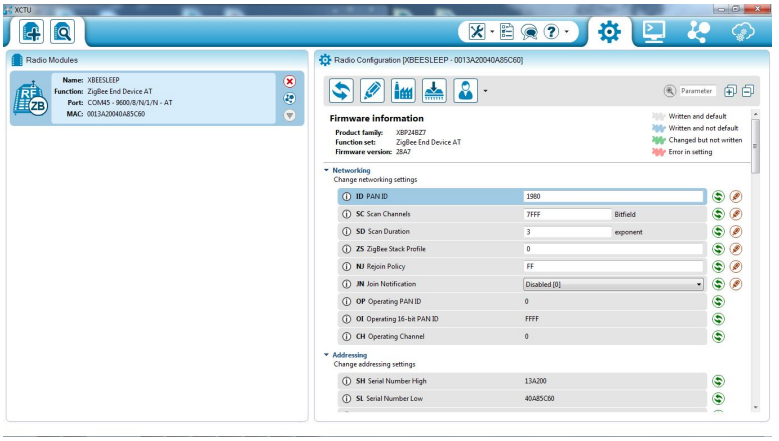
The application will start to look for different devices connected and will deliver a message similar to this one:



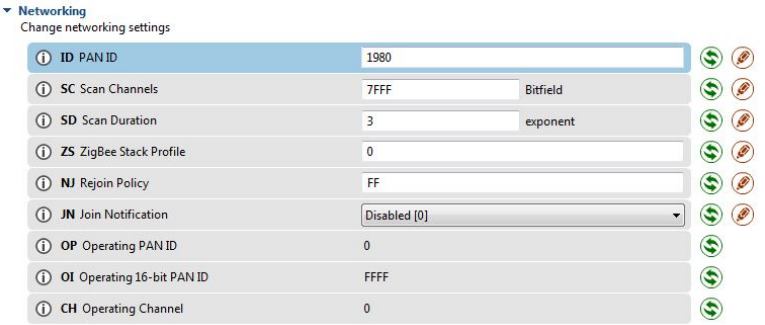
It is possible that X-CTU asks you to do a reset of the XBee in this step (or in the next steps). To do that, just press the reset button in the adapter for 1 or 2 seconds.



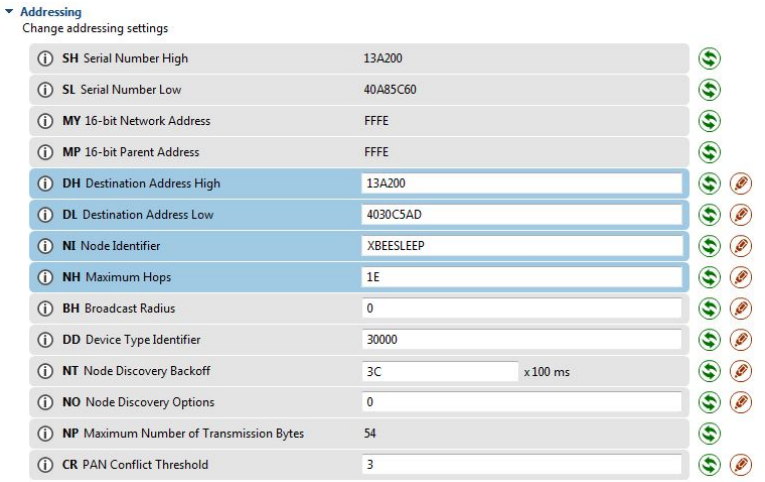
Go to the Configuration Working mode and click to select the device. We must check each parameter first.



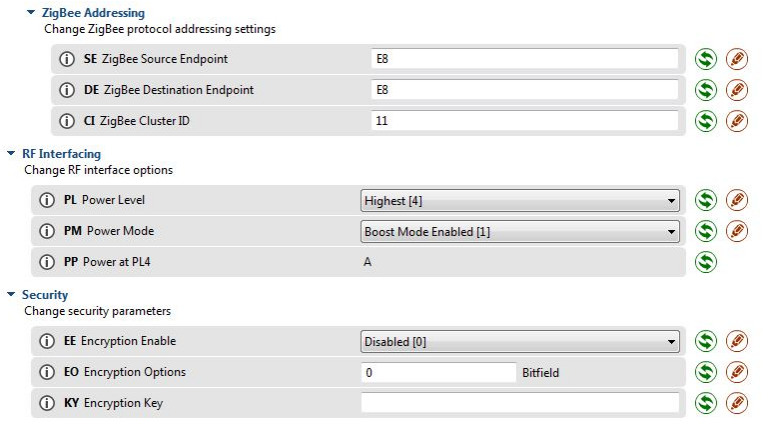
Check the **PAN ID**:



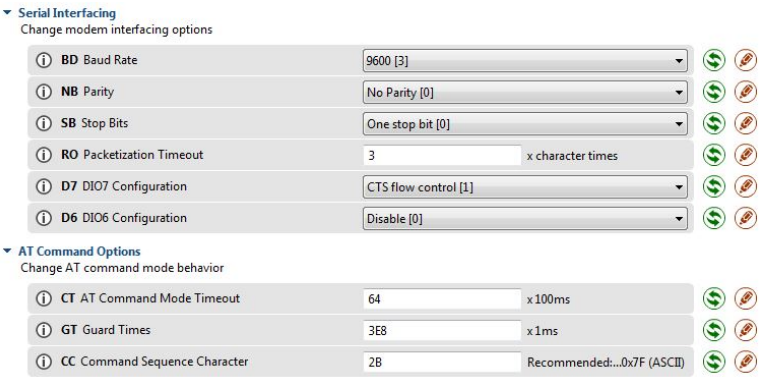
The destination address (2 parts):



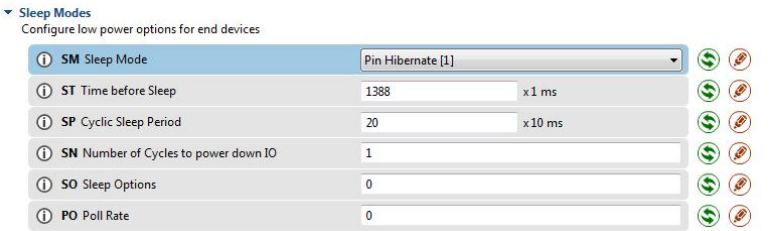
The KY parameter (if needed). It must be set as hexadecimal key:



The serial interface baud rate:



The sleep options for end devices:



In ZigBee End Devices/Routers: the JV command (join verification) enables the power-on join verification check. If enabled, the XBee will attempt to discover the 64-bit address of the coordinator when it first joins a network. If JV=0, the router will continue operating on its current channel even if a coordinator is not detected. If you set to 1 the JV parameter and write it, after rebooting the radio module, it will verify the Coordinator (if it has been configured) is on its operating channel when joining or coming up from a power cycle. If a coordinator is not

detected, the router will leave its current channel and attempt to join a new PAN. This feature can be useful when several ZigBee End Device or Router have not been configured yet; this way you can get them connected to an existing network in a semi-automatic way.

4.3. Troubleshooting

If you have networking problems, please check these tips:

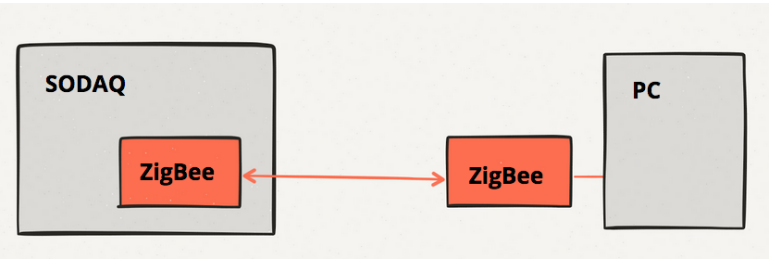
- All XBees are in the same network. Make sure the PAN ID parameter is set.
- Al XBees are in the same channel. The CHANNEL parameter is set with setChannel function. If you want to do it in the X-CTU way, you must use the ATCH command.
- All XBees are configured to the correct baud rate. In X-CTU, you can control data baud rates executing the ATBD command.
- All XBees have the same encryption options. The ENCRYPTION MODE parameter is set with setEncryptionMode function. In X-CTU, the related command is ATEE. And the ENCRYPTION KEY parameter is set with setLinkKey function. In this case, the X-CTU command is ATKY.

5. Sending data between two devices using Zigbee

We will follow the example "Reading Temperature from internal RTC temperature sensor" where we print to the serial port Date, Time, Temperature from RTC and Battery voltage.

Once two radios are configured as explained in the previous chapter, we will connect one to an Mbili board and upload the RTC_date_Volt_Temp_Zigbee.ino code.

The setup is the following:

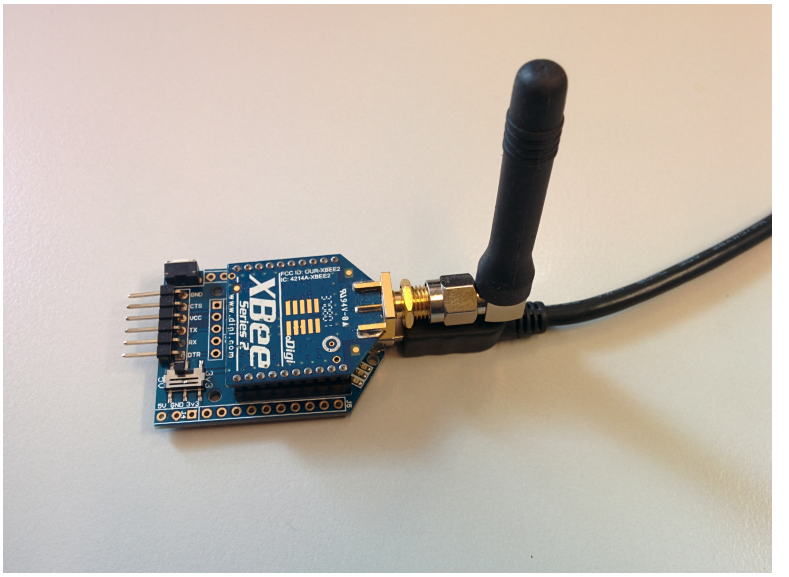


One Zigbee card is connected to the SODAQ board, while the second one is connected to a PC via USB through the UartsBee module.

The only thing to do is print to Serial1 using *Serial1.println(data);* so that data is sent to the Zigbee module instead of the serial cable (as in the original example).

We will also add a sensor number to identify one device from another using *#define DEVICE_NUM 1*

The Zigbee card connected to the PC will look as in the picture below.



The UartsBee is a small board, with a mini USB connector on one side and a Xbee socket on the top. It allows us to communicate with Xbee modules using USB and a serial software.

These is achieved through an Integrated Circuit called FT232RL. Before FT232RL can be used, its drivers must be installed on your Windows based PC from FTDI's website http://www.ftdichip.com. If you are using Ubuntu 12.04, the drivers are already installed in the system.

RTC_date_Volt_Temp_Zigbee.ino

```
//Include the necessary libraries
#include <Wire.h>
#include <Sodaq_DS3231.h>

//Device number
#define DEVICE_NUM 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
```

```
Sodaq_DS3231 RTC; //Create RTC object for DS3231 RTC come Temp Sensor
//year, month, date, hour, min, sec and week-day(starts from 0 and goes to 6)
//writing any non-existent time-data may interfere with normal operation of the RTC.
//Take care of week-day also.*/
DateTime dt(2014, 06, 05, 11, 5, 00, 4);

void setup()
{
  //Start serial
  Serial.begin(9600);
  Serial.println("Date, Time, Temperature, Voltage");

  //Start the I2C protocol
  Wire.begin();

  //Initialise the DS3231
  RTC.begin();
  //remember to comment this line once RTC is updated
  RTC.setDateTime(dt); //Adjust date-time as defined 'dt' above
}

void loop()
{
  ///Read the temperature
  RTC.convertTemperature();
  float temp = RTC.getTemperature();

  // Convert temperature voltage to string
  char buffer[14]; //make buffer large enough for 7 digits
  String temperatureS = dtostrf(temp, 7,2,buffer);
  //'7' digits including '-' negative, decimal and white space. '2' decimal places
  temperatureS.trim(); //trim whitespace, important so Ubidots will treat it as a
 number

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

  String data= DEVICE_NUM +",";
  data += getDateTime()+ ", ";
  data += String(temperatureS)+ "C, ";
  data += String(mv)+ "mV";
  Serial.println(data);
  Serial1.println(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;
}

float getRealBatteryVoltage()
{
  uint16_t batteryVoltage = analogRead(BATVOLTPIN);
  return (ADC_AREF / 1023.0) * (BATVOLT_R1 + BATVOLT_R2) / BATVOLT_R2 *
 batteryVoltage;
}
```

6. Connecting an Analog Sensor

In this section we will describe how the Mbili can send us data from an external Grove analog sensor. We will use Serial Monitor to view the sensor data.

Disconnect the USB cable, and hook up one of the Grove analog sensors to your Mbili (Grove analog temperature sensor and Grove potentiometer are shown below).

Table 5. Connecting an analog sensor example

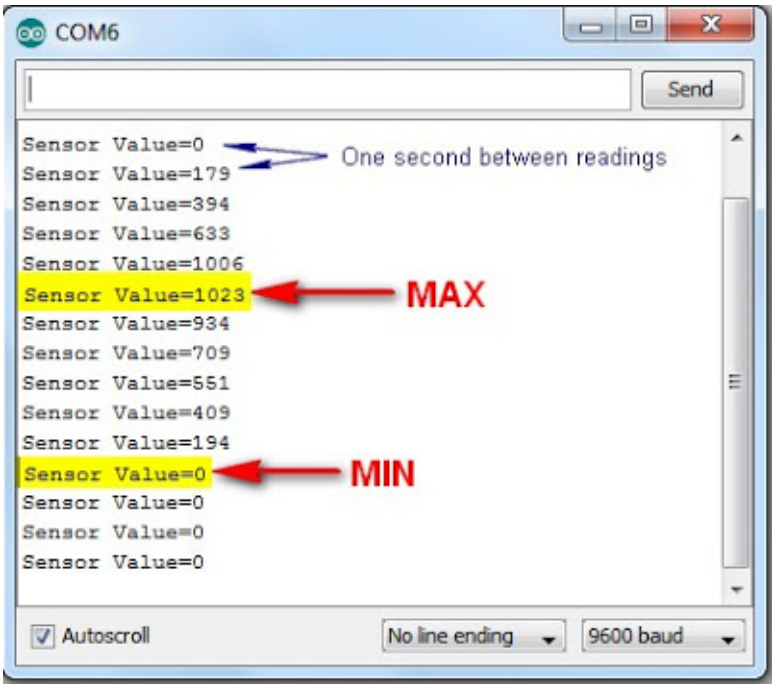
Required Components	SODAQ Mbili Board, Analog Sensor
Required Libraries	None
Hardware Setup	Connect the analog sensor to port A0
Source Code	**SensorAD0.ino**

SensorAD0.ino

```
//Send Sensor Value to Serial Monitor

  int sensorVal = 0;

void setup() {
```

```
  // Setup Serial communication with computer
    Serial.begin(9600);
}

void loop() {
 // Read the value from the sensor:
    sensorVal = analogRead(A0);

 // Send the value to the Serial Monitor
    Serial.print("Sensor Value=");
    Serial.println(sensorVal);

 // Interval between readings = 1 second
    delay(1000);
  }
```

Open the Serial monitor and watch the readings change depending on the input conditions. As an example, by turning the potentiometer from left to right, you get an output similar to the picture below.



As per the Arduino reference site, AnalogRead [17] returns an integer between 0 and 1023. You can see this is true based on the picture above. But what if we do not want a value between

[17] http://arduino.cc/en/Reference/analogRead

0 and 1023. Let us say we want a value between 0 and 100? You would have to use the map function[18]. We will do it by changing line 13 to this:

```
13  sensorVal = map(analogRead(A0),0,1023,0,100);
```

The map function is quite a useful function, and good fun to play around with. So here are some things to try.

Change line 13 to the following, upload to the Arduino and then open the Serial Monitor to see the effect.

Trial 1:

```
13  sensorVal = map(analogRead(A0),0,1023,100,0);
```

Trial 2:

```
13  sensorVal = map(analogRead(A0),0,1023,0,1000);
```

Trial 3:

```
13  sensorVal = map(analogRead(A0),200,800,0,100);
```

In **Trial 1**: We see that the values have been inverted. Instead of ranging from 0 up to100, they now go from 100 down to 0.

In **Trial 2**: The analog readings are now mapped to a range of 0 up to 1000.

In **Trial 3**: The analog readings that range from 200 to 800 are mapped to a range of 0 to 100. Therefore if the analog readings drop below 200, we will end up with a negative value for sensorVal. If the analog readings go above 800, we will end up with a value greater than 100. For this particular example, my readings actually range from -33 to 137.

Therefore an

1. Analog reading of 0 = -33
2. Analog reading of 200 = 0
3. Analog reading of 800 = 100

[18] http://arduino.cc/en/Reference/Map

4. Analog reading of 1023 = 137

What if we don't want the output to go beyond our intended limits of 0 to 100?Then you would have to use the constrain function[19]. This essentially trims the reading range of the sensor, and sets a minimum and maximum value.

Replace line 13 with the following code:

```
13  sensorVal = constrain(map(analogRead(A0),200,800,0,100),0,100);
```

Therefore an

1. Analog reading of 0 = 0
2. Analog reading of 100 = 0
3. Analog reading of 200 = 0
4. Analog reading of 800 = 100
5. Analog reading of 955 = 100
6. Analog reading of 1023 = 100
7. Analog values between 200 and 800 will produce a result between 0 and 100.

7. Using a Potentiometer to Measure an Angle

In this example we will demonstrate how to use an analog potentiometer and how to display the reading in the Serial Monitor. This is another example of using the analog pins[20] for input and using the serial connection to send data to a USB connected PC.

Additionally, this example demonstrates the use of the board reference voltage for processing analog input signals into useable readings. In this case the angular reading expressed in degrees. This type of sensor can be used to measure and record the direction that the wind is blowing.

Table 6. Connecting an potentiometer example

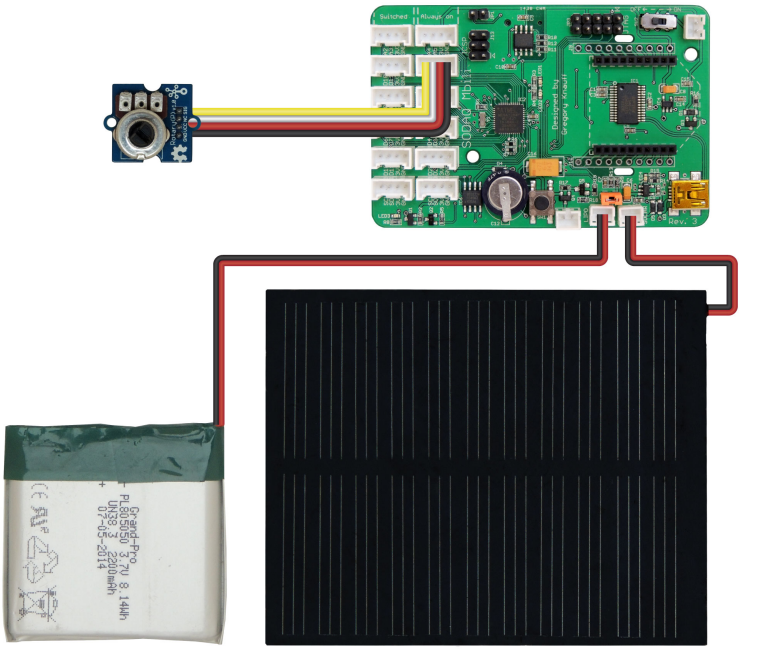
Required Components	SODAQ Mbili Board, Grove Rotary Angle Sensor
Source Code	**Pote_angle.ino**

[19] http://arduino.cc/en/Reference/Constrain
[20] http://arduino.cc/en/Tutorial/AnalogInputPins

Required Libraries	None
Hardware Setup	Connect the Grove Rotary Angle Sensor to port A0
Source Code	**Pote_angle.ino**

You should refer to both the board diagram[21] and Grove sockets page[22] for additional information.

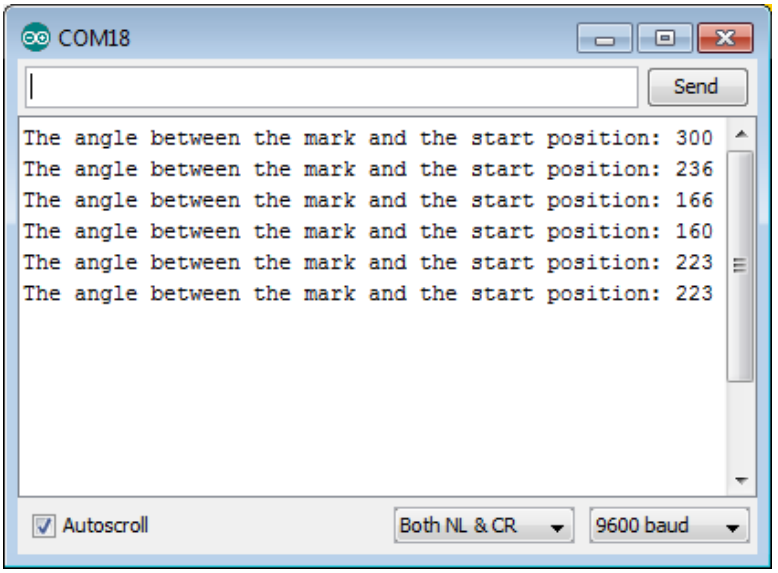
Plug the Rotary Angle Sensor into the socket for the analog pin **A0 A1**. Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets as shown below.



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 opening the Serial Monitor (Ctrl-Shift-M), if you rotate the Rotary Angle Sensor you should see output similar to this:

[21] http://mbili.sodaq.net/?page_id=13
[22] http://mbili.sodaq.net/?page_id=81



Pote_angle.ino

```
#define ROTARY_ANGLE_SENSOR A0 //Use analog pin A0 for the Rotary Angle Sensor
#define ADC_REF 3.3 //Rreference voltage of ADC is 3.3v
#define FULL_ANGLE 300.0 //Full value of the rotary angle is 300 degrees

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

void loop()
{
  //Read the value of the rotary angle sensor in degrees
  int degrees = getDegrees();

  //Output it to the serial monitor
  Serial.print("The angle between the mark and the start position: ");
  Serial.println(degrees);

  //The delay between readings
  delay(500);
}

int getDegrees()
{
  //Read the raw sensor value
```

```
  int sensor_value = analogRead(ROTARY_ANGLE_SENSOR);

  //Convert the sensor reading to degrees and return that value
  float voltage = (float)sensor_value * ADC_REF / 1023;
  float degrees = (voltage * FULL_ANGLE) / ADC_REF;
  return degrees;
}
```

- Here we specify the analog pin which we will connect to the Rotary Angle Sensor. Additionally, we specify several constants, including reference voltages for the ADC and the angular range of the sensor, which are used for converting the analog signal to a value in degrees.

```
#define ROTARY_ANGLE_SENSOR A0 //Use analog pin A0 for the Rotary Angle Sensor
#define ADC_REF 3.3 //Rreference voltage of ADC is 3.3v
#define FULL_ANGLE 300.0 //Full value of the rotary angle is 300 degrees
```

- Here we simply start the serial connection with a call to Serial.begin()[23].

```
void setup()
{
  //Start the serial connection
  Serial.begin(9600);
}
```

- Here we first get the reading from the sensor by calling the user defined method *getDegrees()*. This method returns the processed analog signal as an *integer* value in degrees. We then write data to the outgoing stream buffer of the serial connection using the Serial.print()[24] and Serial.println()[25] methods. This data includes a text description and the reading from the Rotary Angle Sensor in degrees.

```
void loop()
{
  //Read the value of the rotary angle sensor in degrees
  int degrees = getDegrees();

  //Output it to the serial monitor
```

[23] http://arduino.cc/en/Serial/begin
[24] http://arduino.cc/en/Serial/Print
[25] http://arduino.cc/en/Serial/Println

```
  Serial.print("The angle between the mark and the starting position: ");
  Serial.println(degrees);

  //The delay between readings
  delay(500);
}
```

- In this method the raw analog signal is read from the pin connected to the Rotary Angle Sensor. The reading is then processed and returned as a value representing the angle of the sensor in degrees.

First the raw signal is read from the pin using analogRead()[26]. This is then converted into a voltage reading by multiplying it by the *ADC_REF* reference voltage and dividing it by the upper boundary of the reading's range (1023). The result is a floating point value with a range of 0.0…ADC_REF. The final value, in degrees, is calculated by multiplying the reading voltage by the angular range of the sensor (*FULL_ANGLE*) and then dividing it by the*ADC_REF* reference voltage. The final value has a range of 0…FULL_ANGLE as is return by the method.

```
int getDegrees()
{
  //Read the raw sensor value
  int sensor_value = analogRead(ROTARY_ANGLE_SENSOR);

  //Convert the sensor reading to degrees and return that value
  float voltage = (float)sensor_value * ADC_REF / 1023;
  float degrees = (voltage * FULL_ANGLE) / GROVE_VCC;
  return degrees;
}
```

8. Connecting Digital Sensors

In this example we will demonstrate the use of a Switch, with its HIGH and LOW settings. In addition we demonstrate how to use a Relay as an actuator.

This example is a demonstration of the use of the digital pins[27] for both input and output. A digital signal is received from the Switch and is used to activate and deactivate the Relay. The Relay in this example does not serve any real purpose, but if an electronic device such as a coffee-machine was connected to its screw terminal, it could be used to power on and off that device.

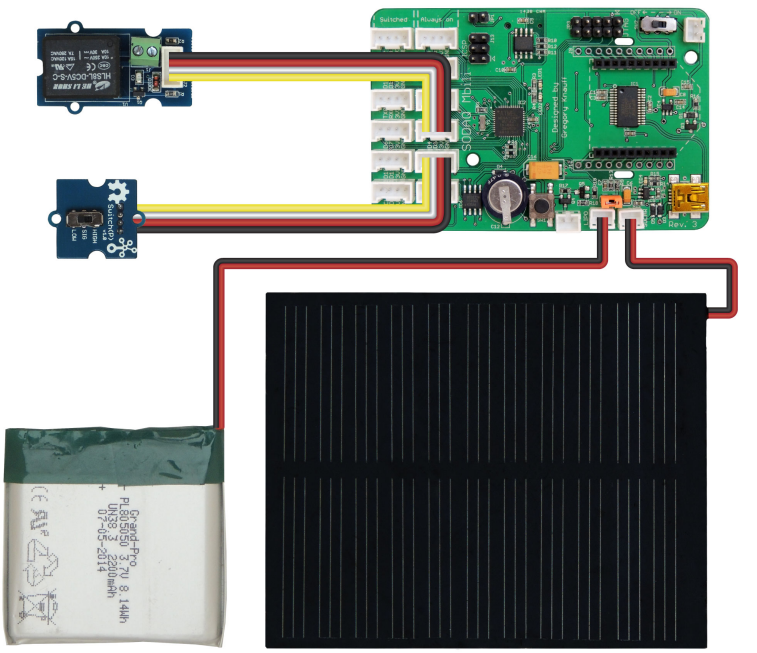
[26] http://arduino.cc/en/Reference/analogRead
[27] http://arduino.cc/en/Tutorial/DigitalPins

Table 7. Connecting Digital Sensors example

Required Components	SODAQ Mbili Board, Grove Relay, Grove Switch
Required Libraries	None
Hardware Setup	Connect the Relay to digital pins **D4 D5**, Connect the Switch to digital pins **D20 D21**
Source Code	**Switch_relay.ino**

You should refer to both the board diagram[28] and Grove sockets page[29] for additional information.

Plug the Relay into the socket for the digital pins **D4 D5**. Plug the Switch into the Grove socket for the digital pins **D20 D21**. 3. Next, plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets. Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE onto the SODAQ Mbili board, and then unplug the USB cable from the computer when it has completed the upload.

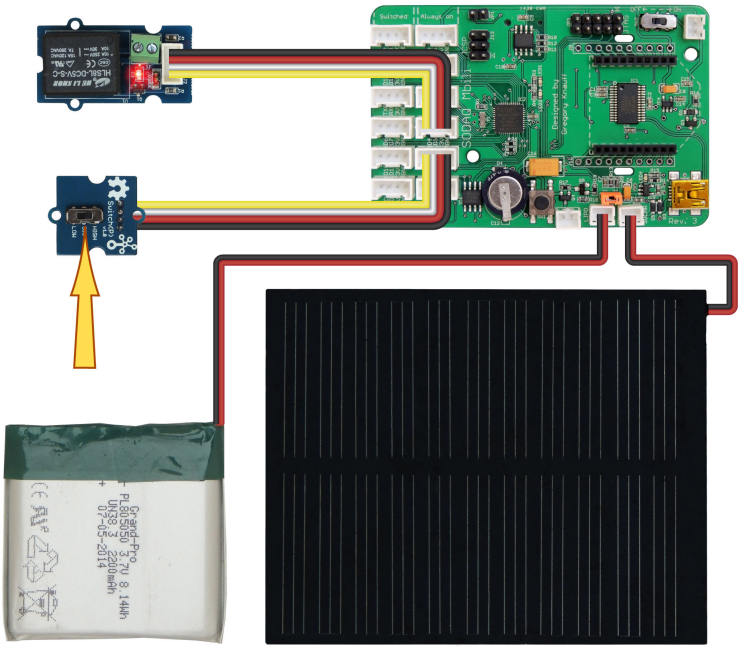


[28] http://mbili.sodaq.net/?page_id=13
[29] http://mbili.sodaq.net/?page_id=81

Switch_relay.ino

```
#define SWITCH_PIN 20 //Use digital pin 20 for the switch
#define RELAY_PIN 4 //Use digital pin 4 for the relay
int switchState = 0;

void setup()
{
  //Set the digital pin modes
  pinMode(SWITCH_PIN, INPUT);
  pinMode(RELAY_PIN, OUTPUT);
}

void loop()
{
  //Read the current state of the switch
  switchState = digitalRead(SWITCH_PIN);

  if (switchState == HIGH)
  {
    //If the switch is set to HIGH, turn the relay on
    digitalWrite(RELAY_PIN, HIGH);
    delay(100);
  }
  else
  {
    //If not, turn the relay off
    digitalWrite(RELAY_PIN, LOW);
  }
}
```

You should see the light on the relay turning on when you the Switch is set to HIGH, and turning off when you set the Switch to LOW.



- Here we specify the digital pins we will use for both the Relay and the Switch. Additionally, we declare a global variable, *switchState*, of type int[30] to store the switch state. This is initialised to 0.

```
#define SWITCH_PIN 20 //Use digital pin 20 for the switch
#define RELAY_PIN 4 //Use digital pin 4 for the relay
int switchState = 0;
```

- Here we set the pin mode for the two digital pins we are using. This is done through a call to pinMode()[31] with the first parameter specifying the pin to be set and the second parameter specifying the mode for that pin. The Switch pin is set to INPUT[32] mode while the Relay pin is set to OUTPUT[33] mode.

```
void setup()
{
  //Set the digital pin modes
```

30 http://arduino.cc/en/Reference/int
31 http://arduino.cc/en/Reference/pinMode
32 http://arduino.cc/en/Reference/Constants
33 http://arduino.cc/en/Reference/Constants

```
  pinMode(SWITCH_PIN, INPUT);
  pinMode(RELAY_PIN, OUTPUT);
}
```

- Here we read the state of the Switch by reading a value from the digital pin that it is connected to. This is done using the digitalRead()[34] method which returns a value matching the built in constants of HIGH[35] or LOW[36]. We then modify the state of the output pin connected to the Relay using the digitalWrite()[37] method, passing the value returned from the Switch. An if/else[38] conditional is used to determine whether to switch the Relay on or off. After switching the relay on, a small delay of 100ms is added (using delay()[39]) to allow the relay activate properly.

```
void loop()
{
  //Read the current state of the switch
  switchState = digitalRead(SWITCH_PIN);

  if (switchState == HIGH)
  {
    //If the switch is set to HIGH, turn the relay on
    digitalWrite(RELAY_PIN, HIGH);
    delay(100);
  }
  else
  {
    //If not, turn the relay off
    digitalWrite(RELAY_PIN, LOW);
  }
}
```

9. Connecting the Digital Sensor TPH using I2C

In this example we will demonstrate the use of the Grove Temperature Pressure Humidity (TPH) Sensor board. The code reads data from the sensor and sends it to the Serial Monitor. We will also demonstrate using the DS3231 Real Time Clock to provide date and time readings.

34 http://arduino.cc/en/Reference/digitalRead
35 http://arduino.cc/en/Reference/Constants
36 http://arduino.cc/en/Reference/Constants
37 http://arduino.cc/en/Reference/digitalWrite
38 http://arduino.cc/en/Reference/Else
39 http://arduino.cc/en/reference/delay

9.1. The SPI Protocol

The Serial Peripheral Interface (SPI) is a synchronous serial data protocol used by the SODAQ Mbili for communication with the MicroSD device and the Serial Flash. On other Arduino microcontrollers it is used for communication with a variety of peripherals and can also be used for communication between two microcontrollers.

9.2. Grove TPH Sensor Board

The Grove TPH Sensor board is a I^2C component which comprises of two separate sensor devices. The first is the SHT21 Sensor which provides temperature and humidity readings. The other is a BMP180 Sensor which provides a second temperature reading as well as a pressure reading.

Table 8. Connecting TPH Sensor example

Required Components	SODAQ Mbili Board, TPH Grove sensor, 0.5W Solar Panel, 1aH Battery Pack
Required Libraries	Wire, Sodaq_BMP085, Sodaq_SHT2x, Sodaq_DS3231
Hardware Setup	Plug the TPH Sensor into the Grove I^2C socket. Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets.
Source Code	**Digital_TPH.ino**

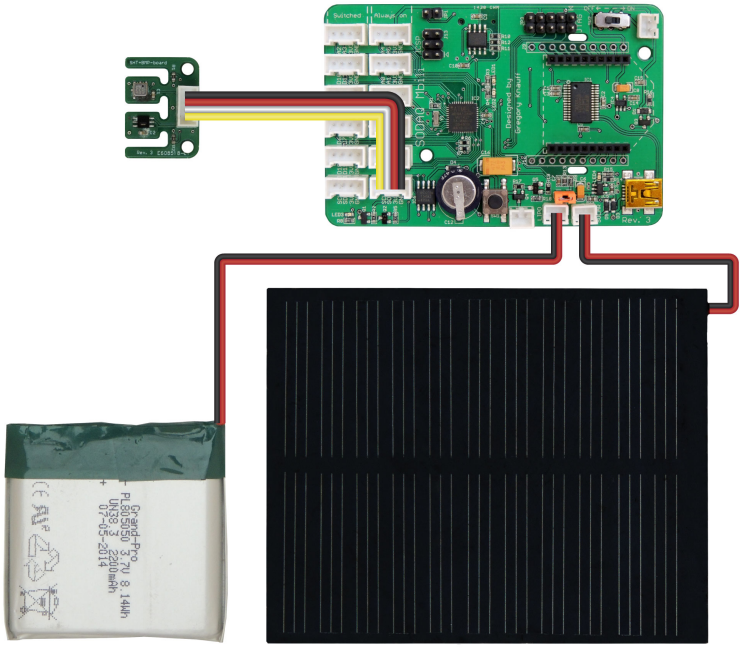
The *Wire* Libraries come pre-installed with the Arduino IDE, and so there is no need to download or install either of them. The*Sodaq_BMP085*, *Sodaq_SHT2x*, and the *Sodaq_DS3231* libraries are included with the SODAQ Mbili files that you have already installed.

If necessary, refer to Section 2[40] of the Getting Started[41] guide for details on where to download from and how to install the SODAQ Mbili files.

You should refer to both the board diagram[42] and Grove sockets page[43] for additional information.

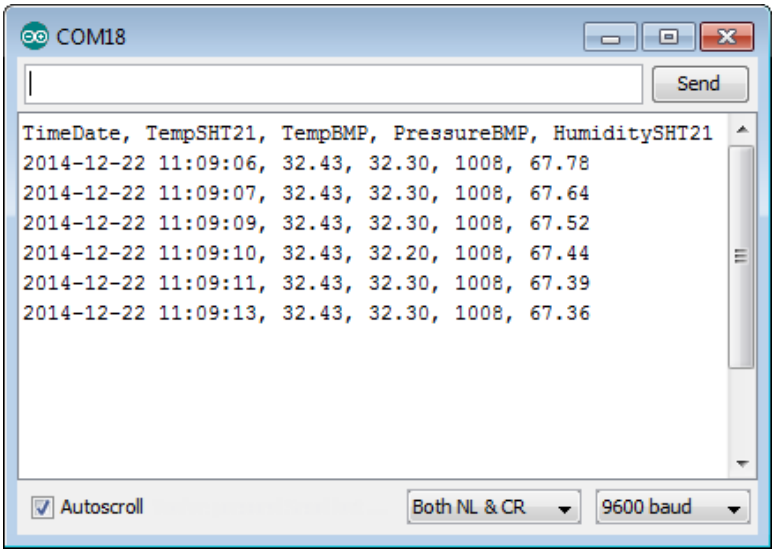
[40] http://mbili.sodaq.net/?page_id=23#step2
[41] http://mbili.sodaq.net/?page_id=23
[42] http://mbili.sodaq.net/?page_id=13
[43] http://mbili.sodaq.net/?page_id=81

Plug the TPH Sensor into the Grove I^{2}C socket. Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets.



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:



Digital_TPH.ino

```
#include <Wire.h>

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

//The delay between the sensor readings
#define READ_DELAY 1000

//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);

  //Initialise sensors
  setupSensors();

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

```
void loop()
{
  //Create the data record
  String dataRec = createDataRecord();

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

  //Wait before taking the next reading
  delay(READ_DELAY);
}

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;
}
```

- Here the necessary library files are included in the sketch using the #include[44] compiler directive.

```
#include <Wire.h>

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

- Here we define the delay between sensor readings and the data header for the log file. We also declare a *Sodaq_BMP085* object.

Note: There is already a global object SHT2x which is used for interfacing with the SHT21 device on the TPH board.

```
//The delay between the sensor readings
#define READ_DELAY 1000

#define DATA_HEADER "TimeDate, TempSHT21, TempBMP, PressureBMP, HumiditySHT21"

//TPH BMP sensor
Sodaq_BMP085 bmp;
```

- Here we start by initialising the serial connection with a call to Serial.begin()[45]. Next we call several user defined methods, one to initialise the sensors we will be using . We will be displaying the sensor data in the Serial Monitor. For readability, we first send the header information to the Serial Monitor with a call to the Serial.println()[46] method passing the parameter *DATA_HEADER*.

```
void setup()
{
  //Initialise the serial connection
```

[44] http://arduino.cc/en/Reference/Include
[45] http://arduino.cc/en/Serial/Begin
[46] http://arduino.cc/en/Serial/Println

```
  Serial.begin(9600);

  //Initialise sensors
  setupSensors();

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

- Here we start by getting the sensor readings with a call to the user defined method *createDataRecord()* which returns us a String[47] containing the current sensor readings. We send the data to the Serial Monitor using the Serial.println()[48] method. Finally, we call delay()[49] to wait *READ_DELAY* number of milliseconds before returning from the *loop()*[50] method. This roughly controls the frequency of the sensor readings.

```
void loop()
{
  //Create the data record
  String dataRec = createDataRecord();

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

  //Wait before taking the next reading
  delay(READ_DELAY);
}
```

- Here we make the necessary calls in order to setup the sensors we will be using. The TPH Sensor communicates via the I^2C protocol and so we must start with a call to Wire.begin()[51]. Next we initialise the *Sodaq_BMP085* Sensor (part of the TPH Sensor board) with a call to *bmp.begin()*. Finally, we initialise the Real Time Clock (RTC) on the DS3231 chip with a call to *rtc.begin()*.

Note: The SHT2x does not require initialisation.

```
void setupSensors()
{
```

[47] http://arduino.cc/en/Reference/string
[48] http://arduino.cc/en/Serial/Println
[49] http://arduino.cc/en/reference/delay
[50] http://arduino.cc/en/Reference/loop
[51] http://arduino.cc/en/Reference/WireBegin

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

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

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

- Here we create and return a String[52] which contains the data readings from the sensors in comma separated format (CSV). The String[53] contains the Time & Date which is queried using the user defined method *getTimeDate()*. We then append the sensor data, using the String += operator[54], from each of the four sensors with a comma separator between each reading. The complete String[55] containing all the data is then returned from this method.

```
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;
}
```

- Here we return a Date & Time reading from the RTC in a String[56] format. First a DateTime[57] object is constructed using a time reading from the DS3231 RTC. This is then converted into a String[58] using the method *DateTime.addToString()* the result of which is then returned from this method.

```
String getDateTime()
```

52 http://arduino.cc/en/Reference/string
53 http://arduino.cc/en/Reference/string
54 http://arduino.cc/en/Reference/StringAppend
55 http://arduino.cc/en/Reference/string
56 http://arduino.cc/en/Reference/string
57 http://playground.arduino.cc/Code/DateTime
58 http://arduino.cc/en/Reference/string

```
{
  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;
}
```

9.3. Using a Button to Activate a Buzzer

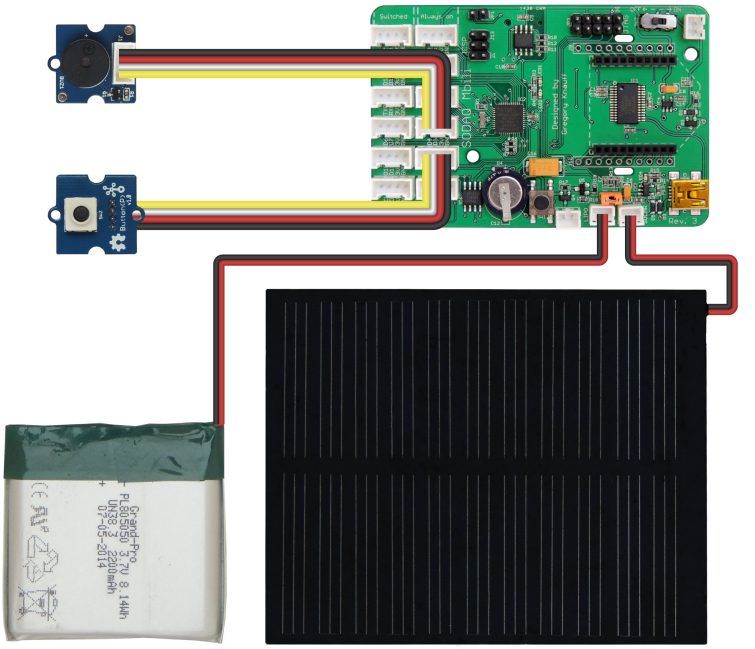
In this example we will demonstrate how to use a Button Sensor and how to create a sound with a Buzzer. This example is a good demonstration of using the digital pins[59] for both input and output. A digital signal is received from the Button Sensor and is used to activate and deactivate the Buzzer.

Table 9. Using a Button to Activate a Buzzer Example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack, Grove Buzzer, Grove Button
Required Libraries	none
Hardware Setup	Plug the Buzzer into the socket for digital pins **D4 D5**. Plug the Button Sensor into the Grove socket for digital pins **D20 D21**. Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets.
Source Code	**Lab2.9.ino**

Plug the sensors as per picture below.

[59] http://arduino.cc/en/Tutorial/DigitalPins



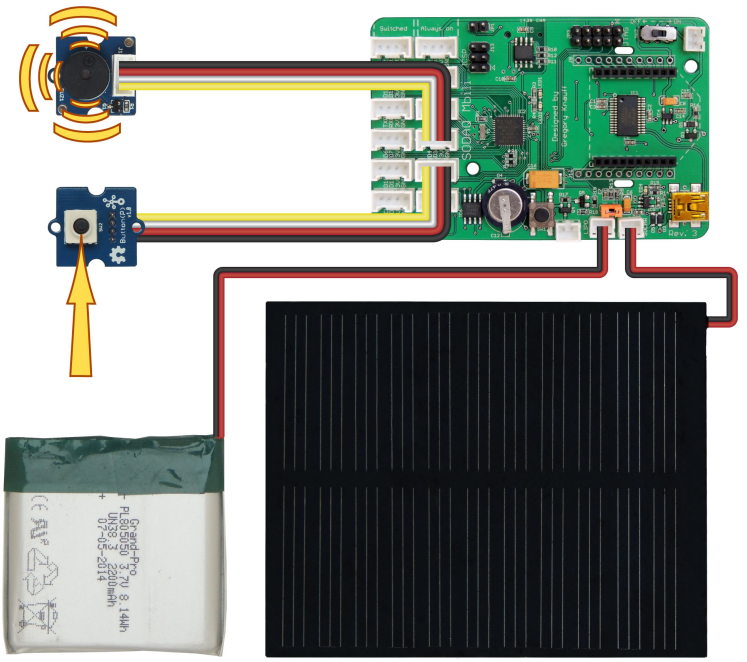
Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE onto the SODAQ Mbili board, and then unplug the USB cable from the computer when it has completed the upload.

Lab2.9.ino

```
#define BUTTON_PIN 20 //Use digital pin 20 for the button
#define BUZZER_PIN 4 //Use digital pin 4 for the buzzer
int buttonState = 0;

void setup()
{
    //Set the digital pin modes
    pinMode(BUZZER_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
}

void loop()
{
    //Read the current state of the button
    buttonState = digitalRead(BUTTON_PIN);
    if (buttonState == HIGH)
    {
        //If the button is pressed, turn the buzzer on
```

```
        digitalWrite(BUZZER_PIN, HIGH);
    }
    else
    {
        //If not, turn the buzzer off
        digitalWrite(BUZZER_PIN, LOW);
    }
}
```

When you press the button, you should hear the buzzer sound.



- Here we specify the digital pins we are using for both the Button Sensor and the Buzzer. Additionally, we declare a global variable *buttonState* and initialise it to 0.

Note: If you use other Grove sockets for either the button or buzzer you must update these values to the first digital pin listed for that Grove socket.

```
#define BUTTON_PIN 20 //Use digital pin 20 for the button
#define BUZZER_PIN 4 //Use digital pin 4 for the buzzer
int buttonState = 0;
```

- Here we set the pin mode for the two digital pins we are using. This is done through a call to pinMode()[60] with the first parameter specifying the pin to be set and the second parameter specifying the mode for that pin. The Buzzer pin is set to OUTPUT[61] mode while the Button pin is set to INPUT[62] mode.

```
void setup()
{
    //Set the digital pin modes
    pinMode(BUZZER_PIN, OUTPUT);
    pinMode(BUTTON_PIN, INPUT);
}
```

- Here we read the state of the Button Sensor by reading a value from the digital pin that it is connected to. This is done using the digitalRead()[63] method which returns a value matching the built in constants of HIGH[64] or LOW[65]. We then modify the state of the output pin connected to the Buzzer using the digitalWrite()[66] method, passing the value returned from the Button Sensor. An if/else[67] conditional is used to determine whether to switch the Buzzer on or off.

```
void loop()
{
    //Read the current state of the button
    buttonState = digitalRead(BUTTON_PIN);
    if (buttonState == HIGH)
    {
        //If the button is pressed, turn the buzzer on
        digitalWrite(BUZZER_PIN, HIGH);
    }
    else
    {
        //If not, turn the buzzer off
        digitalWrite(BUZZER_PIN, LOW);
    }
}
```

60 http://arduino.cc/en/Reference/pinMode
61 http://arduino.cc/en/Reference/Constants
62 http://arduino.cc/en/Reference/Constants
63 http://arduino.cc/en/Reference/digitalRead
64 http://arduino.cc/en/Reference/Constants
65 http://arduino.cc/en/Reference/Constants
66 http://arduino.cc/en/Reference/digitalWrite
67 http://arduino.cc/en/Reference/Else

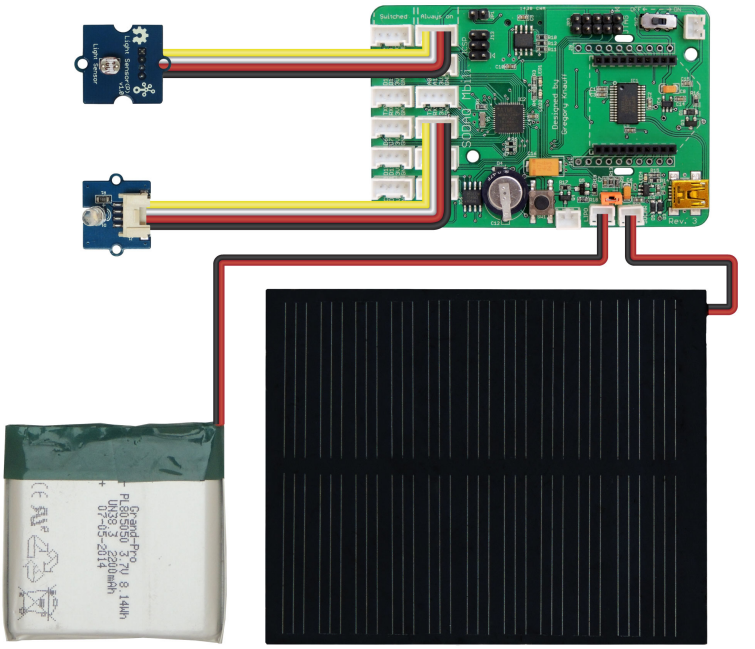
9.4. Turning ON a Light When It Gets Dark

In this example we will demonstrate how to use a Grove Light Sensor to control an LED. The LED will be automatically switched on or off depending on the level of light hitting the sensor. This is a good example of how to use analog input to control a digital output. In this case an analog signal is processed and converted to a digital signal based on a threshold value.

Table 10. Turning ON a Light When It Gets Dark Example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack, Grove Light Sensor, Grove LED (any colour)
Required Libraries	none
Hardware Setup	Plug the Light Sensor into the socket for the analog pins **A4 A5**. Plug the LED into the Grove socket for the digital pins **D4 D5**. Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets.
Source Code	**Lab2.10.ino**

Plug the sensors as per picture below.



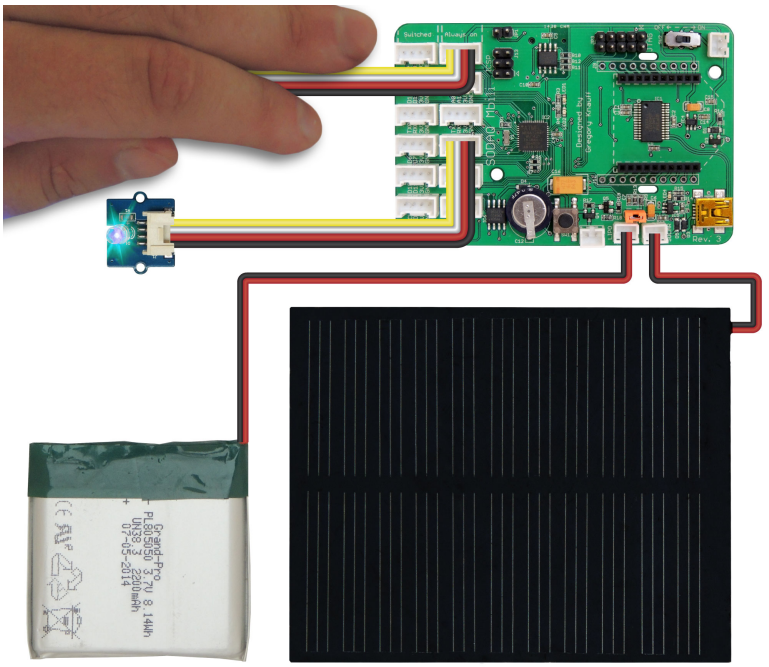
Turn on the SODAQ Mbili board, compile and upload the following sketch from the Arduino IDE onto the SODAQ Mbili board, and then unplug the USB cable from the computer when it has completed the upload.

Lab2.10.ino

```
#define LED_PIN 4 //Use digital pin 4 for the LED
#define SENSOR_PIN A4 //Use analog pin A4 for the sensor
#define THRESHOLD_VALUE 50 //Activation threshold

void setup()
{
  //Set the LED digital pin to OUTPUT mode
  pinMode(LED_PIN, OUTPUT);
}

void loop()
{
  //Read the analog value from the sensor
  int sensorValue = analogRead(SENSOR_PIN);

  //Calculate the resistance from the sensor
  float rSensor=(float)(1023-sensorValue)*10 / sensorValue;
```

```
  //Compare the calculated resistance against the threshold
  if (rSensor > THRESHOLD_VALUE)
  {
    //If the result is above the threshold, turn the LED on
    digitalWrite(LED_PIN, HIGH);
  }
  else
  {
    //If not, turn the LED off
    digitalWrite(LED_PIN, LOW);
  }
}
```

If you cover the Light Sensor with your hand, you should see the LED light up.



- Here we define which pins will be used for both the LED and the Light Sensor. Additionally, we specify an activation threshold for the digital signal. If the processed analog signal is above this threshold then the digital signal (and the LED) is activated.

```
#define LED_PIN 4 //Use digital pin 4 for the LED
#define SENSOR_PIN A4 //Use analog pin A4 for the sensor
#define THRESHOLD_VALUE 50 //Activation threshold
```

- In the setup method we set the specified LED digital pin to OUTPUT[68] mode (using the pinMode()[69] method). There is no need to specify the usage mode for the analog pin connected to the Light Sensor.

```
void setup()
{
  //Set the LED digital pin to OUTPUT mode
  pinMode(LED_PIN, OUTPUT);
}
```

- The first step is to read the raw analog value from the sensor pin. This uses a call to analogRead()[70] which returns a 10 bit unsigned integer value which has a range of 0… 1023. The value is then converted to a logarithmic floating point value representing the resistance across the sensor. This resistance value is compared to the threshold value, if it is above the threshold the digital pin connected to the LED is set to HIGH[71] otherwise it is set to LOW[72] (using the digitalWrite()[73] method).

```
void loop()
{
  //Read the analog value from the sensor
  int sensorValue = analogRead(SENSOR_PIN);

  //Calculate the resistance from the sensor
  float rSensor=(float)(1023-sensorValue)*10 / sensorValue;

  //Compare the calculated resistance against the threshold
  if (rSensor > THRESHOLD_VALUE)
  {
    //If the result is above the threshold, turn the LED on
    digitalWrite(LED_PIN, HIGH);
  }
  else
  {
    //If not, turn the LED off
    digitalWrite(LED_PIN, LOW);
  }
```

68 http://arduino.cc/en/Reference/Constants
69 http://arduino.cc/en/Reference/pinMode
70 http://arduino.cc/en/Reference/analogRead
71 http://arduino.cc/en/Reference/Constants
72 http://arduino.cc/en/Reference/Constants
73 http://arduino.cc/en/Reference/digitalWrite

```
}
```

9.5. Using an Ultrasonic Ranger to Measure Distance

In this example we will demonstrate how to use an Ultrasonic Ranger to measure the distance of an object placed in front of the sensor and to display that distance reading in the Serial Monitor. This is another example of using the digital pins[74] for input and output and for using the serial connection to send data to a USB connected PC. Additionally, we demonstrate the use of single digital pin both in INPUT[75] and OUTPUT[76] modes. This example also uses the pulseIn()[77] method which measures the duration (in µs) of a digital signal received over a digital pin. This time value is then converted into a representative distance in centimetres and is displayed in the serial monitor. Additionally, a LED is lit up whenever the reading falls below a specified threshold.

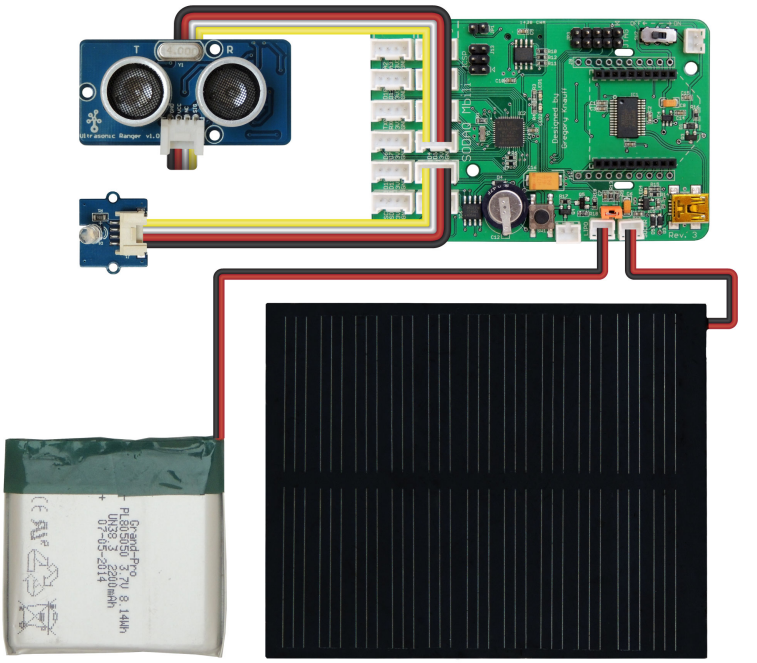
Note: This example is based on the Arduino tutorial that can be found here: Ping Ultrasonic Range Finder Tutorial[78]

Table 11. Using an Ultrasonic Ranger to Measure Distance Example

Required Components	SODAQ Mbili Board, 0.5W Solar Panel, 1aH Battery Pack, Grove Ultrasonic Ranger, Grove LED (any colour)
Required Libraries	none
Hardware Setup	Plug the Ultrasonic Ranger into the socket for the digital pins **D4 D5**. Plug the LED into the Grove socket for the digital pins **D20 D21**. Plug the 0.5W solar panel and the 1A LiPo battery into their respective sockets.
Source Code	**Lab2.11.ino**

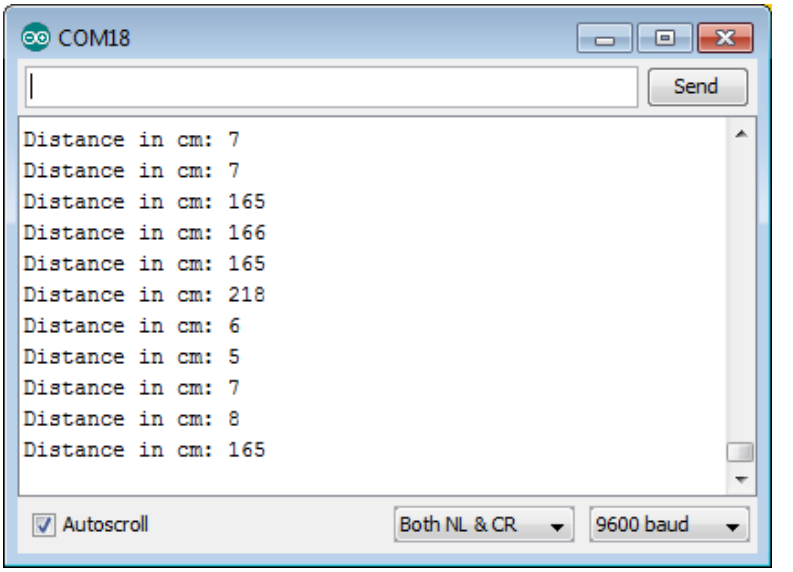
Plug the sensors as per picture below.

[74] http://arduino.cc/en/Tutorial/DigitalPins
[75] http://arduino.cc/en/Reference/Constants
[76] http://arduino.cc/en/Reference/Constants
[77] http://arduino.cc/en/Reference/pulseIn
[78] http://www.arduino.cc/en/Tutorial/Ping



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), if move your hand over the Ultrasonic Ranger you should see output similar to this:

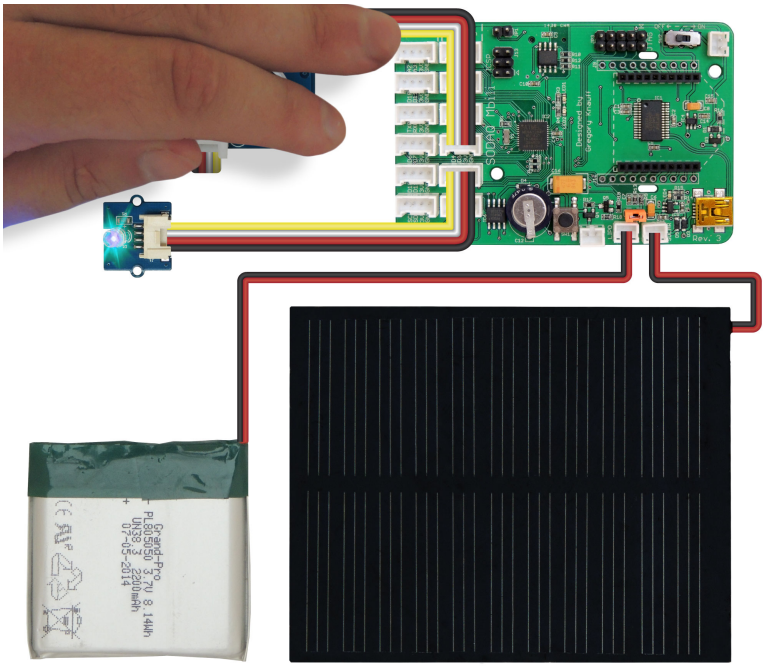


Lab2.11.ino

```
#define PING_PIN 4 //Use digital pin 4 for the range finder
#define LED_PIN 20 //Use digital pin 20 for the LED
#define LED_LIGHT_DISTANCE 5 //At this distance (in cm) or less the LED is lit up
#define READ_DELAY 100 //Delay between readings in milliseconds

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

  //Set the digital pin modes
  pinMode(LED_PIN, OUTPUT);
}

void loop()
{
  //Set Ping Pin to OUTPUT mode
  pinMode(PING_PIN, OUTPUT);

  //Send a HIGH signal for 5 microseconds
  digitalWrite(PING_PIN, HIGH);
  delayMicroseconds(5);
  digitalWrite(PING_PIN, LOW);

  //Now read back the distance as a time value (microseconds) using the same pin
  pinMode(PING_PIN, INPUT);
  long duration = pulseIn(PING_PIN, HIGH);

  //Convert the time into a distance
  long cm = microsecondsToCentimeters(duration);

  //Output the distance to the serial monitor
  Serial.println("Distance in cm: " + String(cm));

  if (cm < LED_LIGHT_DISTANCE)
  {
    //If the distance reading is equal to or less than
    //LED_LIGHT_DISTANCE switch the LED on
    digitalWrite(LED_PIN, HIGH);
  }
  else
  {
    //If not, turn the LED off
    digitalWrite(LED_PIN, LOW);
  }
```

```
  //Delay before the next reading
  delay(READ_DELAY);
}

long microsecondsToCentimeters(long microseconds)
{
  //The speed of sound is 340 m/s or 29 microseconds per centimeter
  //The ping travels twice the distance, there and back and so divide by 58
  return microseconds / 58;
}
```

Additionally, the LED will light up if your hand or another object is within 5 cm of the sensor.



- Here we specify which digital pins to use for both the Ultrasonic Ranger and the LED. Additionally, we specify the threshold distance at which the LED will be lit up, and the delay between taking readings.

```
#define PING_PIN 4 //Use digital pin 4 for the range finder
#define LED_PIN 20 //Use digital pin 20 for the LED
#define LED_LIGHT_DISTANCE 5 //At this distance (in cm) or less the LED is lit up
#define READ_DELAY 100 //Delay between readings in milliseconds
```

- Here we start the serial connection with a call to Serial.begin()[79]. Additionally, we set the digital pin we are using for the LED to OUTPUT[80] mode. This is done through a call to pinMode()[81].

Note: We do not at this point set the pin mode for the Ultrasonic Ranger. This is because we will be using it in both INPUT[82] and OUTPUT[83] modes and we will be switching between those modes in the method loop()[84].

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

  //Set the digital pin modes
  pinMode(LED_PIN, OUTPUT);
}
```

- The Ultrasonic Ranger will activate its emitter whenever it is receiving a HIGH[85] signal over the digital pin that it is connected to. Here we aim to switch on the emitter for a short burst in order to emit a ultrasonic ping.

This can be achieved with several steps. First we must set the connected digital pin to OUTPUT[86] mode using pinMode()[87]. Next we set the digital pin to HIGH[88]. Then we add a short delay using the method delayMicroseconds()[89] which only returns after the specified number of µs have elapsed. We then switch off the emitter by setting its digital pin to LOW[90]. This results in the emitter of the Ultrasonic Ranger being activated for 5µs.

```
void loop()
{
  //Set Ping Pin to OUTPUT mode
```

79 http://arduino.cc/en/Serial/begin
80 http://arduino.cc/en/Reference/Constants
81 http://arduino.cc/en/Reference/pinMode
82 http://arduino.cc/en/Reference/Constants
83 http://arduino.cc/en/Reference/Constants
84 http://arduino.cc/en/Reference/loop
85 http://arduino.cc/en/Reference/Constants
86 http://arduino.cc/en/Reference/Constants
87 http://arduino.cc/en/Reference/pinMode
88 http://arduino.cc/en/Reference/Constants
89 http://arduino.cc/en/Reference/delayMicroseconds
90 http://arduino.cc/en/Reference/Constants

```
pinMode(PING_PIN, OUTPUT);

//Send a HIGH signal for 5 microseconds
digitalWrite(PING_PIN, HIGH);
delayMicroseconds(5);
digitalWrite(PING_PIN, LOW);
```

- The Ultrasonic Ranger measures the time it takes (in µs) for a ultrasonic ping to travel from its emitter, reflect off an object, and return again to its receiver. This travel time can then be read from the Ultrasonic Ranger as a digital pulse signal, with the duration of the digital HIGH[91] pulse matching that of the ultrasonic ping time.

In order to get the reading from the Ultrasonic Ranger, we must first set its digital pin to INPUT[92] mode. We then take a reading using thehttp://arduino.cc/en/Reference/pulseIn[pulseIn()] method which returns the duration of the digital signal it receives from the Ultrasonic Ranger. The reading in µs is converted to a distance value in centimetres using the user defined method *microsecondsToCentimeters()*.

```
//Now read back the distance as a time value (microseconds) using the same pin
pinMode(PING_PIN, INPUT);
long duration = pulseIn(PING_PIN, HIGH);

//Convert the time into a distance
long cm = microsecondsToCentimeters(duration);
```

- Here we write data to the outgoing stream buffer of the serial connection using the Serial.println()[93] method. This data includes a text description and the distance reading from the Ultrasonic Ranger in centimetres. Additionally, an if/else[94] conditional is used to determine whether to switch the LED on or off based on the distance reading and the threshold value *LED_LIGHT_DISTANCE*. Finally, we call the method delay()[95] to wait the specified amount of milliseconds (*READ_DELAY*) before taking the next reading.

```
//Output the distance to the serial monitor
Serial.println("Distance in cm: " + String(cm));

if (cm < LED_LIGHT_DISTANCE)
```

91 http://arduino.cc/en/Reference/Constants
92 http://arduino.cc/en/Reference/Constants
93 http://arduino.cc/en/Serial/Println
94 http://arduino.cc/en/Reference/Else
95 http://arduino.cc/en/reference/delay

```
  {
    //If the distance reading is equal to or less than
    //LED_LIGHT_DISTANCE switch the LED on
    digitalWrite(LED_PIN, HIGH);
  }
  else
  {
    //If not, turn the LED off
    digitalWrite(LED_PIN, LOW);
  }

  //Delay before the next reading
  delay(READ_DELAY);
}
```

- Here the value in microseconds (µs) is converted to the equivalent distance in centimetres (cm). Sound travels approximately 1cm every 29µs. The ultrasonic ping travels twice the distance of the object in front of the sensor, there and back again. So the value returned is that of the parameter value *microseconds* divided by 58 (= 29 * 2).

```
long microsecondsToCentimeters(long microseconds)
{
  //The speed of sound is 340 m/s or 29 microseconds per centimeter
  //The ping travels twice the distance, there and back and so divide by 58
  return microseconds / 58;
}
```

10. Using Temperature, Humidity and Moisture Sensors

10.1. Grove - Moisture Sensor

This Moisture Sensor can be used to measure soil moisture or detect if there is water around the sensor. You can for example let the plants in your garden reach out for human help when they need irrigation.

Once deployed in the ground, when the soil moisture deficits the sensor output value will decrease. You can know whether a plant needs water or not by observing the values that the sensor outputs. The following sketch demonstrates a simple application of sensing the moisture of the soil.

Connect this module to one of analog port A4 of and then insert the Sensor into the soil or place it anywhere you want.

Note: This sensor isn't hardened against contamination or exposure of the control circuitry to water and may be prone to electrolytic corrosion across the probes, so it isn't well suited to being left in place or used outdoors.

Typical values are:

Sensor in air= 0

Sensor in dry soil = 5-50

Sensor in humid soil = around 500

Sensor in water = 900-1000

11. Grove - Temperature and Humidity Sensor Pro

Professional measurements of temperature and relative humidity are possible with this Grove sensor. This is a powerful version of the Grove - Temperature and Humidity Sensor. It has more complete and accurate performance than the basic version. The detecting range of this sensor is 5% RH - 99% RH, and -40°C - 80°C. And its accuracy satisfyingly reaches up to 2% RH and 0.5°C. A professional choice for applications that have relatively strict requirements.

Connect the Temperature and Humidity Sensor Pro to **A0**.

Hum_temp_moist.ino

```
#include "DHT.h"

#define DHTPIN A0     // what pin we're connected to
#define MOISTPIN A4    // select the input pin for the potentiometer


DHT dht(DHTPIN,DHT11);

void setup()
{
    Serial.begin(9600);
    Serial.println("Humidity, Temperature and Moisture");

    dht.begin();
}

void loop()
{
    // Reading temperature or humidity takes about 250 milliseconds!
```

```
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
    float h = dht.readHumidity();
    float t = dht.readTemperature();

    // check if returns are valid, if they are NaN (not a number) then something
 went wrong!
    if (isnan(t) || isnan(h))
    {
        Serial.println("Failed to read from DHT");
    }
    else
    {
        Serial.print(h);
        Serial.print(",");
        Serial.print(t);
        Serial.print(",");
        Serial.println(analogRead(MOISTPIN));

    }
}
```