Design and implementation weather station system based on wireless sensor network

Project Team

Raed AL-Warasneh Rawaa Radaydeh

Shatha AL-Amayreh

Project Supervisor

Prof.Karim Tahboub

Submitted to the College of Engineering

in partial fulfillment of the requirements for the degree

Communication and Electronics and Mechatronics Engineering

Palestine Polytechnic University

Hebron – Palestine

December 2016

Palestine polytechnic University

Hebron – Palestine

College of Engineering

Electrical and Mechanical Engineering Departments

Design and implementation weather station system based on wireless sensor network

Project Team

Raed AL-Warasneh Rawaa Radaydeh

Shatha AL-Amayreh

By the guidance of our supervisor , and by the acceptance of all members in the testing committee , this project is delivered to department of electrical and mechanical engineering in the college of engineering and technology , to be as a partial fulfillment of the requirement of the department for the degree of CE.

Supervisor signature

____

Testing committee signature

__________________________ __________________

The head of department signature

____

جامعة بوليتكنك فلسطين

الخليل – فلسطين

كلية الهندسة

دائرة الهندسة الكهربائية

Design and implementation weather station system based on wireless sensor network

بناء على نظام كلية الهندسة واشراف ومتابعة المشرف المباشر على المشروع وموافقة اعضاء اللجنة المناقشة , تم تقديم هذا العمل الى دائرة الهندسة الكهربائية وذلك للوفاء بمتطلبات درجة البكالوريوس في هندسة الاتصالات والالكترونيات وهندسة الميكاترونيكس

توقيع المشرف

———————————————

توقيع اللجنة المناقشة

————————————   ————————————

توقيع رئيس الدائرة

——————————————————

الاهداء

نـهدي هذا الجهد المتواضع بداية الى من أخرج البشرية جمعاء من الظلمات الى النور محمد صلى الله عليه وسلم

الى امهاتنا وابائنا اللذين وصلوا ليلهم بالنـهار حتى يرونا نكبر يوما بعد يوم

الى اخوتنا واخواتنا اللذين لم ولا يبخلوا علينا بشيء

الى اصداقائنا واحبابنا و اللذين بمحبتهم اناروا الدرب

الى الشموع اللتي تحترق وتضيء للاخرين الدروب أساتذتنا اللذين لم يبخلوا علينا بكل ما لديهم

الى كل من سبقنا و أضاء بعلمه عقل غيره وهدى بالجواب الصحيح حيرة سائليه فأظهر بسماحته تواضع العلماء وبرحابته سماحة العارفين وأخجلنا باهتمامه

الى كل من ساعدنا ولو بجملة او حتى كلمة

الى كل محب للعلم ومتيم به

الى اولئك اللذين حرموا حريتهم خلف القضبان لجل هذا الوطن الغالي

الى اولئك اللذين فقدوا حياتهم لكي نبقى نحن على هذا الوطن ولا نفرط بحبة تراب منه

الى كل من وقعت عينه على هذا العمل المتواضع نـهديكم اياه ونقدمه لكم راجين من المولى ان يتقبله ويجعله في ميزان حسناتنا

Palestine Polytechnic University

College of Engineering and Technology

Electrical and Mechanical Engineering Departments

Communication and Electronics and Mechatronics Engineering

Graduation project

Design and implementation weather station system based on wireless sensor network

Project Team

Raed AL–Warasneh　　　　Rawaa Radaydeh

Shatha AL–Amayreh

Project Supervisor

Prof.Karim Tahboub

Hebron – Palestine
December 2016

Acknowledgments

First of all thanks ALLAH for giving us patience robust minds and will to understand and treat the needs of our world that lead us to completion our graduation project.

We shall express our deep gratitude to those who help us in whole strides of our life and support us every time our mothers fathers sisters brothers families and friends.

We also indebted and deeply grateful to our supervisor and director Prof.Karim Tahboub for his support encouragement and his help to take some advantages of his experience and knowledge this thanks reach to all teachers doctors and supervisors that we met in our preceptorial life from first class in primary stage until now …

الملخص

تقوم فكرة المشروع بشكل عام على بناء نظام محطات أرصاد جوية ،حيث يقوم بإدارة هذه المحطات من ناحية قياس المتغيرات المختلفة للطقس وعرض هذه المتغيرات ونشرها بطريقة منظمة وسريعة وغير مكلفة ، بالإضافة الى ذلك سيوفر المشروع الكثير من الوقت والجهد للمستخدمين الذين يهتمون بمعرفة هذه المتغيرات مقارنة بالطرق التقليدية .

يستخدم المشروع تقنية (GPRS) اللاسلكية التي تدعمها شبكة الهواتف المحمولة (GSM) للقيام بعملية نقل المعلومات من أماكن تواجد هذه المحطات الى الخادم .

كل ذلك سوف يتم التعامل معه عن طريق برنامج مخصص الا وهو (ARDUINO IDE) والداعم لأوامر الاتصال بالمودم (AT commands) والداعمة أيضا لبناء واجهة للمستخدم لتسهيل الحصول على هذه البيانات .

مقابل ذلك سوف تتم قراءة المتغيرات المختلفة للجو عن طريق مجسات موصولة بمعالج (ARDUINO) حيث سيقوم الأخير بجمع هذه المتغيرات والتواصل مع المودم وارسال هذه البيانات الى الخادم لعرضها ونشرها وتخزينها . موفرا بذلك كل الوقت المستخدم لعملية نقل البيانات والجهد المبذول للقيام بذلك .

ABSTRACT

In this project we aim to design weather station in order to provide us with some weather variables every day such as temperature, humidity, amount of rainfall level, barometric pressure and speed and direction of wind and others . The main function for this station is to send data one time every three hours during the day by using the global system for mobile communication (GSM/GPRS) modem which is connected directly with a microcontroller. To make the project more efficient, some parameters are taken into account like cost, availability of components, the project time life, power consumption, accuracy at sending and receiving of data.

the main aim of giving data from different stations is knowing the weather state accurately in any time, and any location. taking the weather information from these stations and disseminate it will satisfy this.

Table of Content

List of Tables

List of Figures

CHAPTER ONE

INTRODUCTION

1.1 Introduction

The weather has always a great influence on people’s health and modes habits, daily activities, productivity, performance and social behavior, etc. Sometimes, the connection is direct and obvious. "Knowing the weather" and trying to forecast it correctly can make a difference for the survival and prosperity of mankind.

Weather data help us make choices. Whether you will need an umbrella when you go to school tomorrow, whether you can cycle to your friend’s house, whether or not you can go to the beach at week’s end. Farming is always affected by the weather no matter what sort of farming it is, the weather will affect it.

With advances in technology in recent years, new methods and equipment have been developed to monitor weather, and collect weather information. With this new technology it is possible to make more accurate measurements for any period of time.

Today, one of the most frequently used type of equipments are wireless weather stations.

1.2 Project Objective

Design two prototypes of remote weather stations(with same design), to read the temperature, humidity, amount of rain fall, barometric pressure, wind speed and direction, forming a base for scalable stations that could absorbs other future nodes(weather stations).

1.3 Project Motivation

1. Palestine as a country, suffers from a significant lack of weather stations to cover the entire country. The biggest motivation is to design and build a modern plant that meets the required purpose.

2. Due to the availability of wireless communication devices and techniques that led us to exploit these technologies and their use in the design and construction of a station.

3. The high cost of import stations manufactured in some of the companies and their access to Palestine led to design and building station.

1.4 Related work

1.4.1 The Weather station

This project was for students in Houston Baptist University it measures temperature, rainfall, humidity, barometric pressure and wind speed, the aim of this project is to compare data gathered from the weather station in 2011 to local weather since 1900, and in the near future include working with Information Technology Services to make up-to-date campus weather conditions available to all University students through portal. Additional plans include installing a lightning rod that will be beneficial for outdoor sporting events and a soil density reader for biology lab, this project shown as figure1.1.

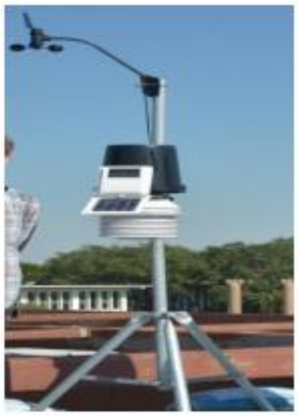


Figure 1.1: the weather station project

1.4.2 Home Made Logger of Temperature and Solar Data[1]

This project made by David Cook, it is designed to be located away from the house without a power cord and it is also Operate for long periods of time without human intervention or maintenance, retain data in the event of a power loss or microcontroller reset. Survive all outdoor temperatures and conditions (such as rain and snow).The weather station consists of a small solar panel (about 1/8 watt), rechargeable battery backup for night time and cloudy operation, microcontroller ,data stored in a 4 MB flash chip, two power source monitors, and six temperature sensors. The six temperature sensors four are taped to a bamboo pole at various heights, one is in the project box, and one is a couple of inches underground. This project shown as Figure 1.2.

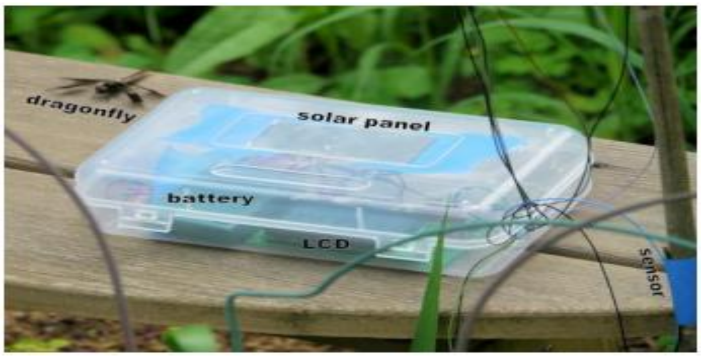


Figure 1.2: Home Made Logger of Temperature and Solar Data

1.4.3 Tenerife Weather Station

The weather station runs 24 hours a day, 7 days a week and 365 days a year; logging data and providing weather status information to many systems within the observatory, The weather station consist of many sensors, wind speed 1, wind speed 2, wind direction, internal temperature, external temperature, rain, dew, cloud south north, east, and west, solar radiation, light sun rise and set, barometric pressure. One function of the weather station is to provide a good or bad signal to the control server, the good or bad signal is generated every 10 seconds; the output is controlled by a series of rules, if any of the rules report that the weather is bad the output signal is set to bad, this information is used to govern the operation of the control server. In addition to the control server the good or bad output is also used by numerous safety systems that act to protect the system in the event of a failure elsewhere in the system. The weather station collects samples from each sensor every 10 seconds; every 10 minutes the weather station calculates average, minimum and maximum values and inserts them into the database, from here the database logs are uploaded to the primary UK server, from which they are used to generate the weather statistics and graphs.

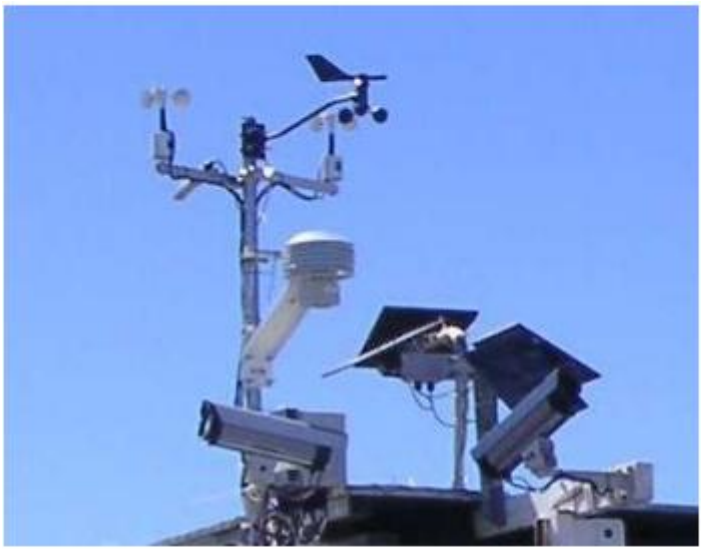


Figure 1.3: Tenerife Weather Station

1.4.4 Weather Station (EasiData Mark 4)[2]

The EasiData Mark 4 is a highly flexible data logger that until 2009 formed the core component of Environdata's modular weather stations. These weather stations can be customized to suit almost any application or location, The EasiData Mark 4 can store up to 216 data types, including real time calculations on sensor inputs (eg. Current sensor readings, means, maximums, minimums, true vector wind analysis, evaporation rate etc). New commands can be sent to the logger at any time to reconfigure its storage operation. The Mark 4 can be programmed to meet specific requirements now, and re-programmed when those requirements change. An Environ data Easi Data Mark 4 Weather Station is a robust computerized system that measures and records environmental conditions in real time. Weather station consists of four key components, external sensors that measure environmental parameters and transmit raw data, A data logger that 9 collects and analyses the raw data and calculates the results before storing it into internal memories, A communications method of the choice that transmits the stored data to any external computer, Environ data's software that will download and display data in preferred format. Environ data manufactures almost 20 different types of sensors to operate with the Easi Data system, These include: solar radiation, relative humidity, air temperature, wet and dry bulb temperature, grass temperature, soil temperature, rainfall, wind speed, wind direction, barometric pressure, soil moisture, radiant heat, ultra violet radiation, and Photo synthetically active radiation. Remote Communications Equipment for project include : GSM Remote Link (Global System for Mobile Communications), CDMA Remote Link

(Code Division Multiple Access),Modem Dial-Up – Via a standard phone line, UHF Radio Link, Network Connection, or RS-232 Serial Cable Link – maximum preferred distance 100 meters, up to 5 km with line drivers.

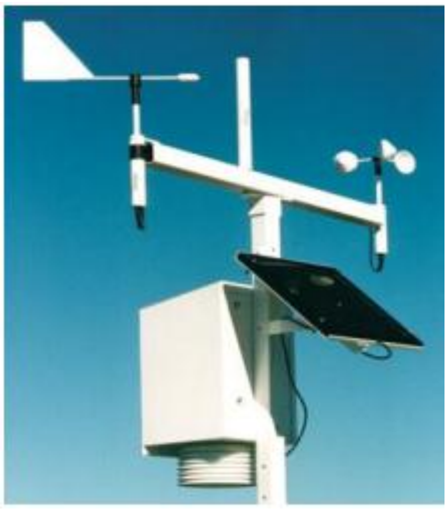


Figure 1.4: Weather Station (Easi Data Mark)

1.4.5 Conclusion

When the group of this project check out the related projects, they found many differences between them. These differences can summaries as the following:

1. The weather station Project does not contain GSM modem to provide the destination by collected data moment by moment .
2. Home Made logger of Temperature and Solar Data project has not GSM for giving data moment by moment, and do not have the ability to provide the other conditions of the weather like wind and direction, and humidity
3. Tenerife Weather Station project needs to connect directly to 8 a source power, so this project not useful to be installed in isolated area to measured weather conditions, and also it is more cost.
4. Weather Station[Easi Data Mark 4] project not contains system for rechargeable battery, so in isolated area not useful, and also it is high cost.

1.5 Time Schedule

Week / Tasks	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
System design	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦								
Development phase									🟦	🟦						
Test phase											🟦	🟦	🟦	🟦		
Documentation														🟦	🟦	🟦

1.6 Cost Analysis

PART	AMOUNT	COST/Piece (NIS)
Arduino Mega	2	175x2
GSM/GPRS Shield	2	250x2
Jawwal SIM	2	300
Monthly pill for Jawwal	-	90x2
DHT11	2	25x2
BMP180	2	25x2
Power bank	2	80x2
Wooden works	-	200x2
Documintation	-	70
Total		2060

1.7 Project Overview

In chapter 2, system design, serves to introduce the technology and related terms used in this thesis. Reason of choosing the sensors of the project.

In chapter 3, talk about GPRS and its feature.

Chapter 4,the project implementation and interfacing between the sensor of the project and Arduino .Chapter five contain the result of testing and operating the project, the final one contain the faced problems and recommendations .

CHAPTER TWO

THEORITICAL BACKGROUND

2.1 General Packet Radio Service(GPRS)

2.1.1 Overview [3]

GPRS is a data network architecture that is designed to integrate with the existing GSM networks and offer mobile subscriber “always on” packet switched data services access to the corporate networks and the internet.

GPRS provides mobile operators with an opportunity to offer higher-margin data access services to subscribers. In return subscribers benefit from GPRS by being able to use higher band width mobile connections to the internet and corporate networks. GPRS tunneling protocol is the protocol used by GSM or UTMS operators to convert radio signals from subscribers into data packets, and then to transport them in non-encrypted tunnels.

With the addition GPRS to GSM, mobile operators are adding mobile internet and virtual private network services to their existing mobile voice services. GPRS networks are connected to several external data networks including those of roaming partners, corporate customers, GPRS roaming exchange providers and the public internet.

GPRS can be thought of as an overlay network onto a second generation GSM network. This data overlay network provides packet data transport at rates from 9.6 to 171 kbps. In addition, multiple users can share the same air-interface resources.

2.1.2 How Does GPRS Work?[4]

When a user turns on a GPRS device, typically it will automatically scan for a local GPRS channel. If an appropriate channel is detected , the device will attempt to attach to the network. The SGSN receives the attach request, fetches subscribers profile information from the subscribers HLR node and authenticates the user. Ciphering may be established at this point.

2.1.3 IP Address [5]

An Internet Protocol address (IP address) is a numerical label assigned to each device (e.g., computer, printer) participating in a computer network that uses the Internet Protocol for communication. [DOD Standard Internet Protocol (January 1980)] An IP address serves two principal functions: host or network interface identification and location addressing. Its role has been characterized as follows: "A name indicates what we seek. An address indicates where it is. A route indicates how to get there.

There are two types of IP addresses:

1. Static IP address, which is a unique and fixed IP address on a network.

2. Dynamic IP address, which is assigned by means of the dynamic host configuration protocol every time a device starts.

2.1.4 Advantages of GPRS [6]

GPRS is a non-voice service that provides wireless packet data access within GSM ‘’Global System for Mobile communication’’ networks. Although newer, faster mobile technologies such as Edge; 3G (Third Generation); Universal Mobile Telecommunication Service; and high-speed download packet access, or HSDPA access have been developed for mobile devices, GPRS is still supported by most mobile networks.

2.2 What is Arduino? [7]

Arduino is an open-source prototyping platform based on easy-to-use hardware and software. Arduino provides an open-source and easy-to-use programming tool, for writing code and uploading it to your board. It is often referred to as the Arduino IDE (Integrated Development Environment).

The Arduino boards are able to read inputs - light, proximity or air quality on a sensor, or an SMS or Twitter message- and turn it into an output- activating a motor, turning on a light, publishing content online or trigger external events. You can tell your board what to do by writing code and uploading it to the microcontroller on it using the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has powered thousands of projects. Arduino has gathered around a community where beginners and experts from around the world share ideas, knowledge and their collective experience. There are thousands of makers, students, artists, designers, programmers, researchers, professionals and hobbyists worldwide who use Arduino for learning, prototyping, and finished professional work production.

Arduino was born at the Interaction Design Institute Ivrea IDII from the Wiring project as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. The main objective of both projects is to make the process of working with technology and electronics easier. The Arduino board has evolved to adapt to new needs ranging from simple 8-bit boards to products ready for IoT applications. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software is open-source, and it is growing through the contributions of developers and the Arduino community worldwide.

There have been many similar projects, but none of them succeeded as well as Arduino has, due to how easy it is to use the software, and the affordability of the hardware. The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users needs. It runs on Mac, Windows, and Linux. This project used Arduino due to inexpensive and flexible hardware, simple programming environment, cross-platform, open source and extensible software Open source and extensible hardware.

2.2.1 Arduino MEGA 2560[8]

The Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

Feature

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 Ma
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
Length	101.52 mm
Width	53.3 mm
Weight	37 g

2.2.2 SIM900 GPRS/GSM Shield[9]

The SIM900 GSM/GPRS Shield provides you a way to use the GSM cell phone network to receive data from a remote location. The shield allows you to achieve this via any of the three methods:

1. Short Message Service
2. Audio
3. GPRS Service

The GPRS Shield is compatible with all boards which have the same form factor (and pin out) as a standard Arduino Board. The GPRS Shield is configured and controlled via its UART using simple AT commands. Based on the SIM900 module from SIMCOM, the GPRS Shield is like a cell phone. Besides the communications features, the GPRS Shield has 12 GPIOs, 2 PWMs and an ADC**.**

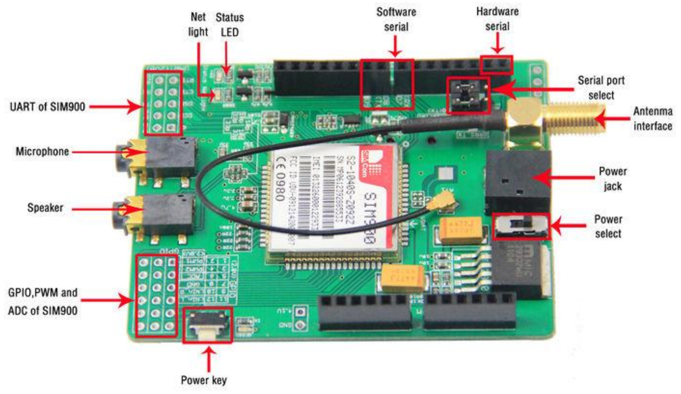


Figure 2.1: GPRS shield form

Feature:

1. Quad-Band 850 / 900/ 1800 / 1900 MHz - would work on GSM networks in all countries across the world.
2. Control via AT commands - Standard Commands: GSM 07.07 & 07.05 | Enhanced Commands: SIMCOM AT Commands.
3. Short Message Service - so that you can send small amounts of data over the network (ASCII or raw hexadecimal).
4. Embedded TCP/UDP stack - allows you to upload data to a web server.
5. Speaker and Headphone jacks - so that you can send DTMF signals or play recording like an answering machine.
6. SIM Card holder and GSM Antenna - present onboard.

7. 12 GPIOs, 2 PWMs and an ADC (all 2.8 volt logic) - to augment your Arduino.
8. Low power consumption - 1.5mA(sleep mode).
9. Industrial Temperature Range - -40°C to +85 °C.

3.3 Programming Language [10]

The Arduino programming language is a simplified version of C/C++. If the operator know C, programming the Arduino will be familiar. If the operator do not know C, no need to worry as only a few commands are needed to perform useful functions. An important feature of the Arduino is that the operator can create a control program on the host PC, download it to the Arduino and it will run automatically. Remove the USB cable connection to the PC, and the program will still run from the top each time you push the reset button. Remove the battery and put the Arduino board in a closet for six months. When the operator reconnect the battery, the last program the operator stored will run. This means that when the operator connect the board to the host PC to develop and debug his program, but once that is done, the operator no longer need the PC to run the program.

3.4 SQL Database[11]

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), it is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

A relational database system contains one or more objects called tables. The data or information for the database are stored in these tables. Tables are uniquely identified by their names and are comprised of columns and rows. Columns contain the column name, data type, and any other attributes for the column. Rows contain the records or data for the columns.

3.5 AT Commands [12]

The Hayes command set is a specific command language originally developed by Dennis Hayes for the Hayes Smart modem 300 baud modem in 1981. The command set consists of a series of short text strings which can be combined to produce commands for operations such as dialing, hanging up, and changing the parameters of the connection.

AT commands are instructions used to control a modem. AT is the abbreviation of Attention. Every command line starts with "AT" or "at". That's why modem commands are called AT commands. Many of the commands that are used to control wired dial-up modems, such as ATD (Dial), ATA (Answer), ATH (Hook control) and ATO (Return to online data state), are also supported by GSM/GPRS modems and mobile phones. Besides this common AT command set, GSM/GPRS modems and mobile phones support an AT command set that is specific to the GSM technology, which includes SMS-related commands like AT+CMGS (Send SMS message), AT+CMSS (Send SMS message from storage), AT+CMGL (List SMS messages) and AT+CMGR (Read SMS messages).Note that the starting "AT" is the prefix that informs the modem about the start of a command line. It is not part of the AT command name. For example, D is the actual AT command name in ATD and +CMGS is the actual AT command name in AT+CMGS. However, some books and web sites use them interchangeably as the name of an AT command.

There are two types of AT commands: basic commands and extended commands; Basic commands are AT commands that do not start with "+". For example, D (Dial), A (Answer), H (Hook control) and O (Return to online data state) are basic commands, Extended commands are AT commands that start with "+". All GSM AT commands are extended commands. For example, +CMGS (Send SMS message), +CMSS (Send SMS message from storage), +CMGL (List SMS messages) and +CMGR (Read SMS messages) are extended commands.

3.5.1 Here are some of the tasks that can be done using AT commands with a GSM/GPRS modem or mobile phone:

1. Get basic information about the mobile phone or GSM/GPRS modem. For example, name of manufacturer (AT+CGMI), model number (AT+CGMM), IMEI number (International Mobile Equipment Identity) (AT+CGSN) and software version (AT+CGMR).

2. Get basic information about the subscriber. For example, MSISDN (AT+CNUM) and IMSI number (International Mobile Subscriber Identity) (AT+CIMI).
3. Get the current status of the mobile phone or GSM/GPRS modem. For example, mobile phone activity status (AT+CPAS), mobile network registration status (AT+CREG), radio signal strength (AT+CSQ), battery charge level and battery charging status (AT+CBC).
4. Establish a data connection or voice connection to a remote modem (ATD, ATA, etc).
5. Send and receive fax (ATD, ATA, AT+F*).
6. Send (AT+CMGS, AT+CMSS), read (AT+CMGR, AT+CMGL), write (AT+CMGW) or delete (AT+CMGD) SMS messages and obtain notifications of newly received SMS messages (AT+CNMI).
7. Read (AT+CPBR), write (AT+CPBW) or search (AT+CPBF) phonebook entries.
8. Perform security-related tasks, such as opening or closing facility locks (AT+CLCK), checking whether a facility is locked (AT+CLCK) and changing passwords (AT+CPWD).
9. Control the presentation of result codes / error messages of AT commands. For example, you can control whether to enable certain error messages (AT+CMEE) and whether error messages should be displayed in numeric format or verbose format (AT+CMEE=1 or AT+CMEE=2).
10. Get or change the configurations of the mobile phone or GSM/GPRS modem. For example, change the GSM network (AT+COPS), bearer service type (AT+CBST), radio link protocol parameters (AT+CRLP), SMS center address (AT+CSCA) and storage of SMS messages (AT+CPMS).
11. Save and restore configurations of the mobile phone or GSM/GPRS modem. For example, save (AT+CSAS) and restore (AT+CRES) settings related to SMS messaging such as the SMS center address.

Note that mobile phone manufacturers usually do not implement all AT commands, command parameters and parameter values in their mobile phones. Also, the behavior of the implemented AT commands may be different from that defined in the standard. In general, GSM/GPRS modems designed for wireless applications have better support of AT commands than ordinary mobile phones.

In addition, some AT commands require the support of mobile network operators. For example, SMS over GPRS can be enabled on some GPRS mobile phones and GPRS modems with the +CGSMS command (command name in text: Select Service for SMS Messages). But if the mobile network operator does not support the transmission of SMS over GPRS, the user cannot use this feature.

CHAPTER THREE

SYSTEM DESIGN

This system has been designed during to the need of the global metrological center, the need of local Palestinian metrological center and the need for a scalable expanded weather station networks, these needs are as the following:

3.1 Recognition of the need

3.1.1 Need of global metrological center

Until the adoption of the readings of the weather variables from each station by the World Meteorological Organization :

It must be the state that you want to send readings to the variables of the weather in the World Meteorological Organization member.

The ability of local meteorological readings on encryption that has been get from the monitoring stations, especially the Global Meteorological Organization language and be in different formats such as BUFR, CREX.

Weather Data takes every three hours Coordinated Universal Time (GMT).

International standards to build a weather station:

1. The Rain gauge of the station is placed at a height of 50 cm to 80 cm from the surface of the earth.
2. Temperature and humidity sensors have to be in a box in order to measure the temperature and humidity properly without external influences ,the Box of these sensors rises from the earth's surface from 1 meter to 2 meters.
3. Pressure sensor have to be placed in a closed box.
4. The stations set in an open area but if it placed in a close area each station must have an empty area that equal three time of the station height.

3.1.2 Need of local Palestinian metrological center

The weather variables that the Palestinian Meteorological is interested to measures are temperature, humidity, atmospheric pressure, wind speed and direction and amount of rainfall and the intensity of the brightness of the sun.

Palestinian Meteorological distributes there stations on Jenin, Tulkarm, Qalqilya, Nablus, doma, Jericho, Kardala, Ramallah, Ni'lin, Bethlehem and Hebron.

These stations are controlled remotely with a center to receive data from all stations by sending data through text messages up to the main center by using a data logger to collection all variables measurement and send these data variables to the main center .

Table 3.1 The highest and lowest value registered climatic element in Palestine .

Variables	Maximum	Minimum
Temperature	+50.5C	-10C
wind speed	57 a node (114 km / h)	0
Relative Humidity	100%	2%
Atmospheric pressure	1064.4(hpa)	848
the rain	The highest amount of daily precipitation(123mm)	0

3.2 The need for scalable expanded weather station networks:

1. Increasing the density of an existing network by providing data from new sites and from sites that are difficult to access and inhospitable regions.
2. Supplying, for manned stations, data outside the normal working hours.
3. Increasing the reliability of measurements by using sophisticated technology and modern, digital measurement techniques.
4. Ensuring the homogeneity of networks by standardizing the measuring techniques.
5. Satisfying new observational needs and requirements.
6. Reducing human errors.
7. Lowering operational costs by reducing the number of observers.

3.3Project Equipment

3.3.1 **Arduino Mega 2560 [13]**

There are many version of Arduino, they are differ in structure and tasks. In this project Arduino mega 2560 is used; it has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a USB connection, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started. The Mega 2560 board is compatible with SIM 900 GSM/GPRS shields.

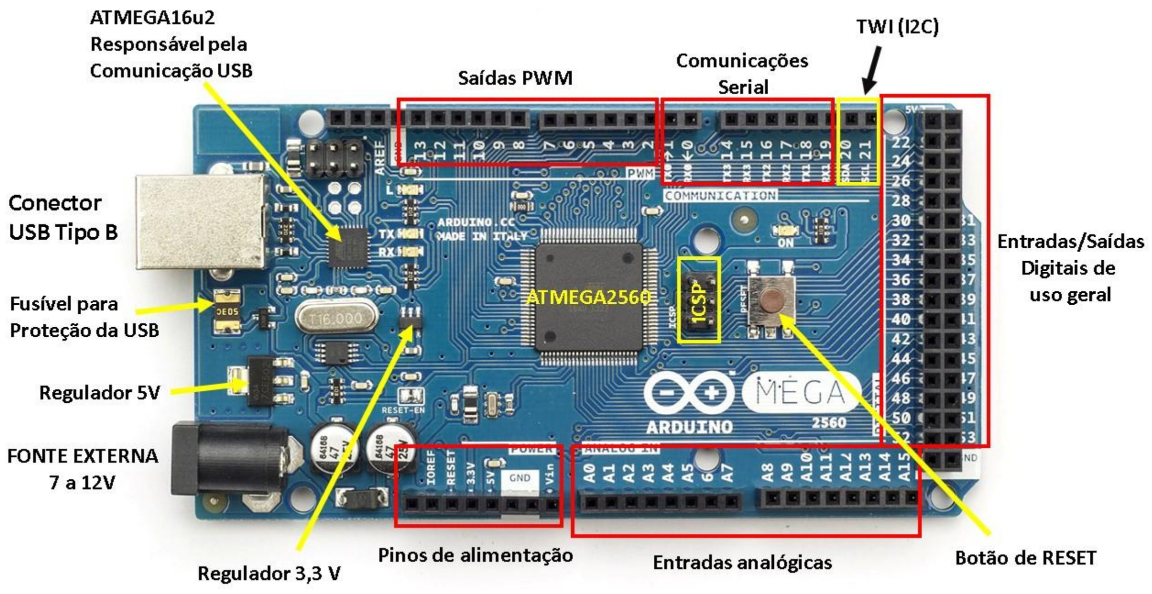


Figure 3.1: Arduino Mega 2560

3.3.2 **SIM 900 GSM/GPRS Shield**

There are different type of GPRS modems , that differs in the interface and module types.

In this project the SIM900GSM/GPRS shield will used due to its capability of supporting the AT commands that are needed to make the connection between the Arduino and the shield; it is used also to communicate with the server. In addition this SIM has an low power consumption.The maximum output current is 450 mille ampere and the maximum output voltage is 5.2 volts which can be used to power some external functions such as sensors.

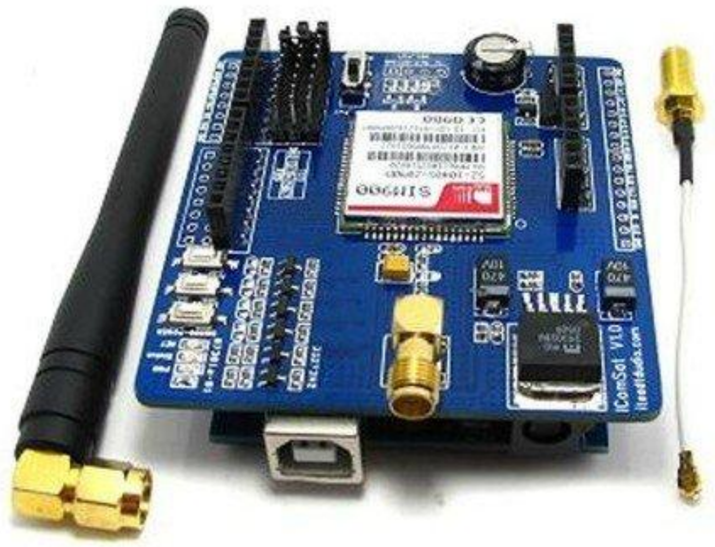


Figure 2.2: SIM900 GSM/GPRS shield

3.3.3 **Temperature and Humidity Sensor**

This project use DHT11 to measure temperature and humidity; that’s because of its availability as first reason, its more accurate than other available temperature and humidity sensors in Palestine and it has low price. Maximum output power is 5.5 volts .

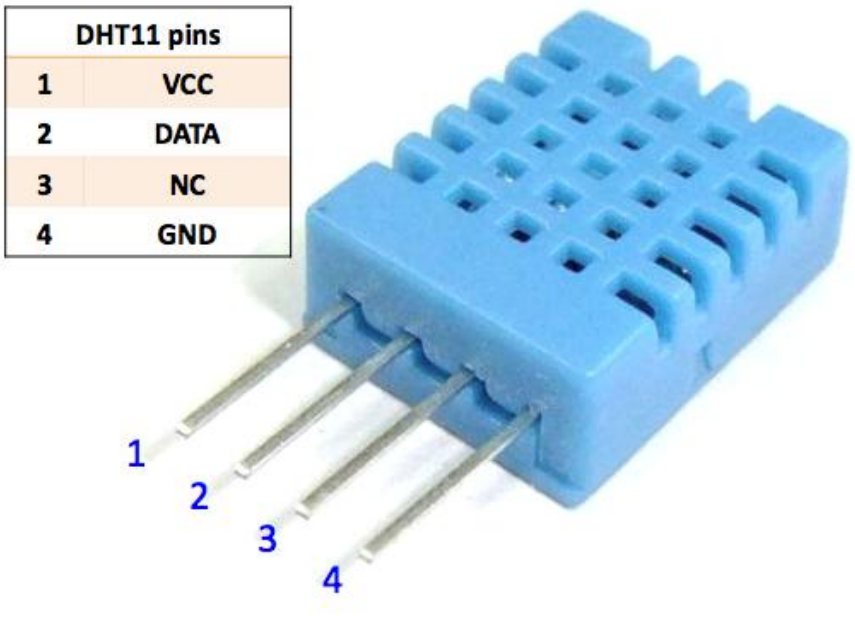

DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND

Figure 2.3: DHT11

3.3.4 **Pressure Sensor**

In this project BMP barometric pressure sensor is used to measure the pressure.

This sensor used for its low price. Maximum output power is 5 volts. It is measure the ranges of the local pressure also.

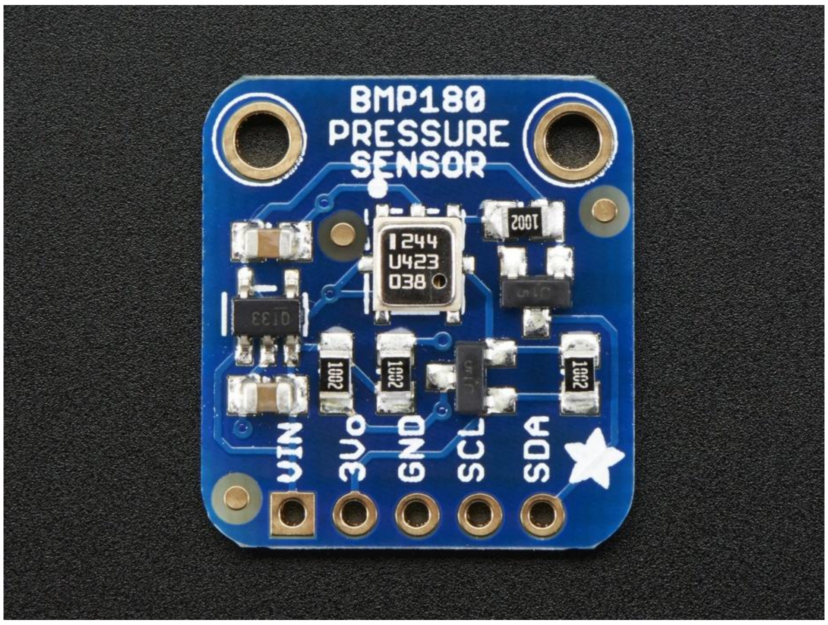


Figure 2.4: BMP180

2.4 Conceptual design

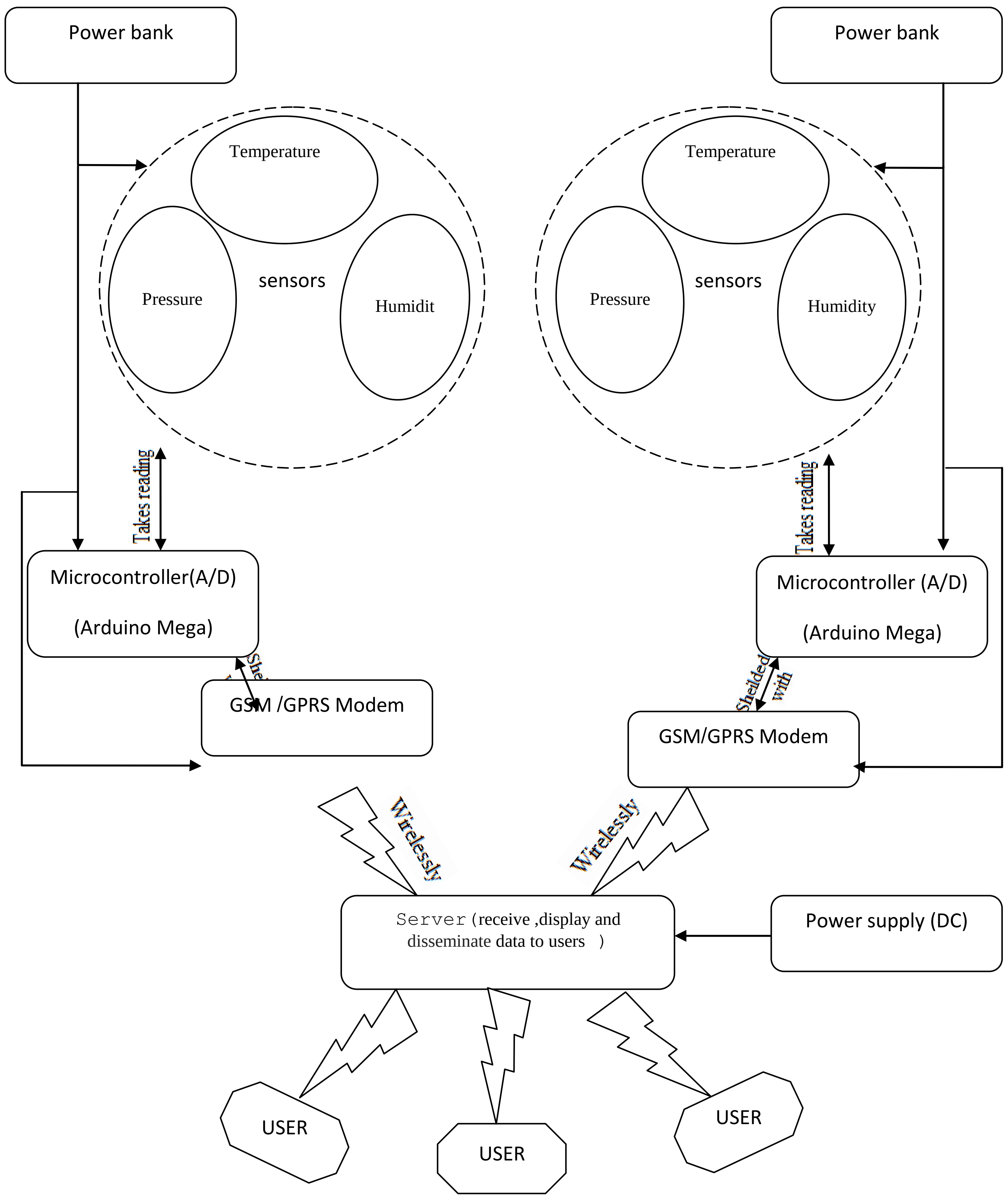


Figure 2.5: conceptual design of weather stations system based on WSN.

The main aim of this design is weather station with scalable property. The conceptual design of a weather station consists of a microcontroller(Arduino) to processing of signals weather parameters. And sensors to measure weather parameter by using temperature, and humidity sensor to treat the temperature and humidity signal, wind speed and direction sensor to measure a speed and direction of the wind-as future sight-, rain gauge sensor to measure the amount of rain fall-as future sight- these sensor enter to the microcontroller that convert the analog signal that come from weather to digital signal to make the connection to the other devices that dealing with digital signal allowable; servers may be used to disseminate the information of each station.

GSM/GPRS connected with microcontroller to send a processing data to main center (Server) that update and storage these data. The component get a power from power source likes DC source(we choose power bank to this tasks) .

The main center (Server) of this design will receive data of all stations save it, choose the appropriate way to display this data and send it to the server in order to facilitate reaching to these information.

2.5 The Server

It's the main point of the system .It will have all instruction that control the displaying and disseminating the data. It will have a special software that will allow the operator to monitor and control the network .The software mainly contain a database and GUI.

The system can be partially controlled from any where.

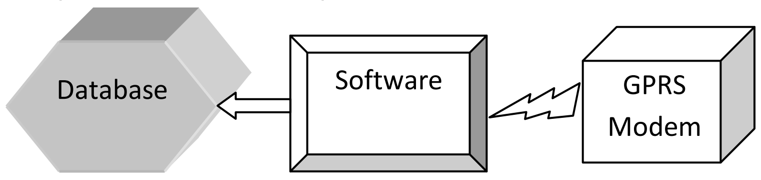


Figure (2.6) the basic component of the Server

2.6 Software Design

The node types in the system are the same .Each one of them has its parameter. The software will be able to identify these parameters of each sensor in all nodes-that distribute in many places-.

2.6.1 The main tasks of the server

One of the main tasks of the software is establishing the connection with GPRS modem at each node; this achieved by AT commands .

This software collect the data about weather variables from all nodes and store it in the corresponding database.

The program gives the users the ability of adding a new nodes with all information that specifies this node to the software when needed (as the following flowchart shown).

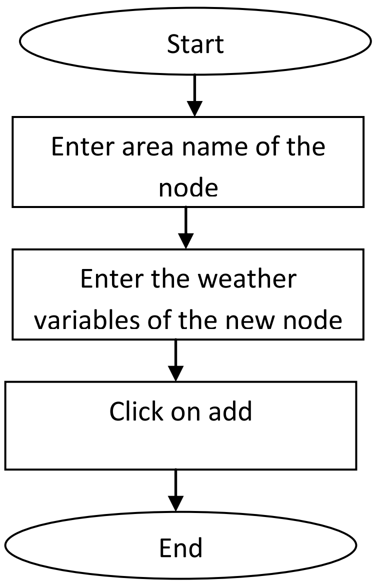

Start
Enter area name of the node
Enter the weather variables of the new node
Click on add
End

Figure 2.7: Adding new node to the system flowchart

2.6.2 Database

A database will be needed as an archive. This archive will have the history of each node that contain the existing weather variables. Adding and updating data will depend on the defined variables at two sides (Arduino and server); the new data will be update the old one. The time and the date will be update with every new reading.

At server the reading of temperature, humidity and pressure are saving as new data in database.

The code of the software of the database will contain some equations over the time and weather variables reading in order to give the users updating data every ten minutes and averaging data every three hours.

There is a specific admin that control the network of the node and the normal users just see the results.

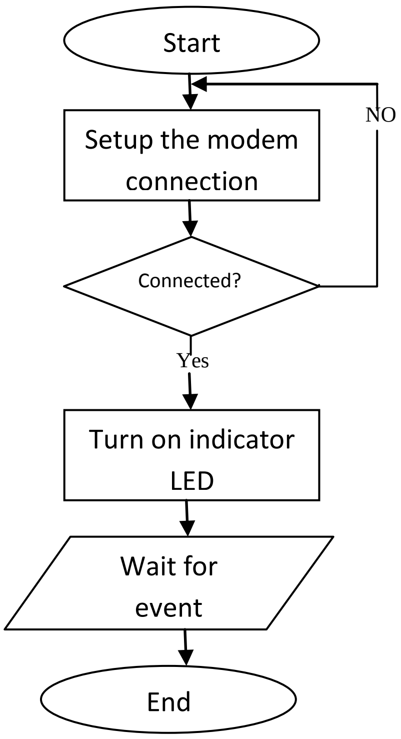
2.6.3 The Microcontrollers Software(Arduino IDE)

The software of the project microcontroller (Arduino IDE) will make the Arduino capable of receiving data from all sensors treat it and send these readings to the server.

As advanced step this software can make any calculations that the operators need.

Each node in the system will have an Arduino that perform different operations depending on the sensors that connecting with it and what the operator want-that depending on the needs of the users-.

When the operator want to add a new node he will need to the following flowchart to connect the Arduino with the Modem.



First of all in each node, the Arduino needs to setup a connection with the GPRS modem by using AT commands the Arduino initiate the connection; if the connection succeeds then a LED indicator will turn on as a notification then the Arduino will wait an instructions or events from modem or sensors.

CHAPTER FOUR

SYSTEM IMPLEMENTATION

4.1 Interfacing Circuits:

Arduino Mega is the main component used to process the signals it receives from the sensors, these signals describes the condition of the weather at each node. To make the Arduino be able to read the input signals correctly and respond accordingly different interfaces circuits will be used to connect these sensors with the Arduino.

Arduino programming language is used to write the different program of different sensors.

4.1.1 Interfacing circuit between Arduino and Temperature & Humidity Sensor (DHT11):

This sensor is used for measure the temperature and the humidity of the weather.

The sensor connected with Arduino as shown in the following circuit :

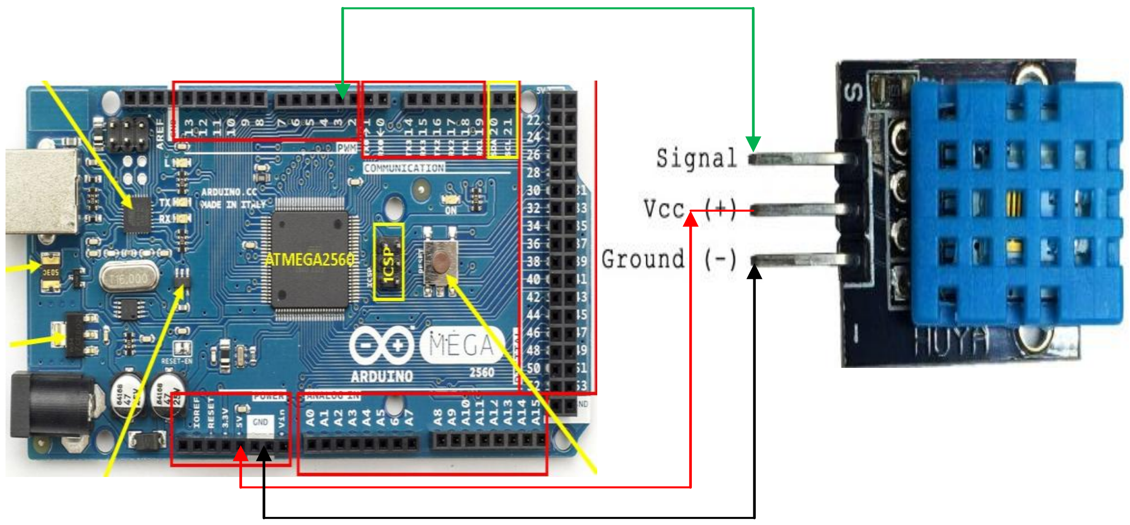


Figure 4.1: The interfacing circuit used to connect Arduino with DHT11

The connection in reality shown below :

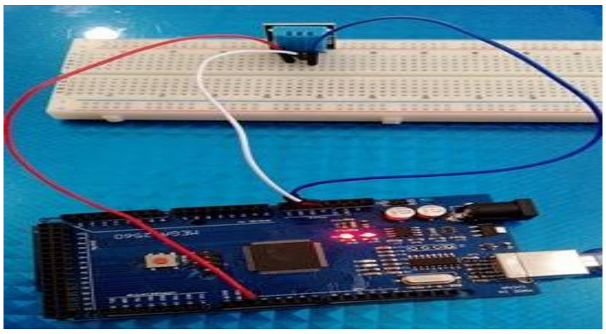


Figure 4.2: Arduino with DHT11

The code of interfacing between Arduino and DHT11-it is used for testing also- is :

4.1.2 Interfacing circuit between Arduino and Pressure Sensor (BMP180)

This sensor is used for measure the pressure of the weather.

The BMP180 sensor has been connected to the Arduino as shown in the following circuit:

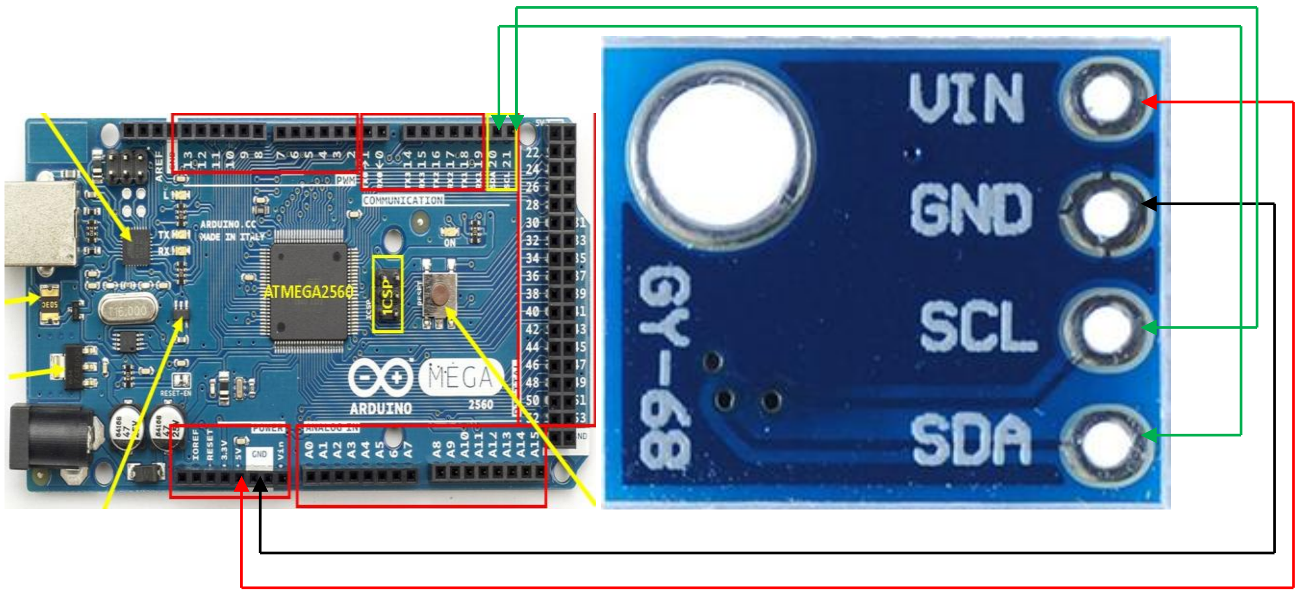


Figure 4.3: The interfacing circuit used to connect Arduino with BMP180

The connection in reality shown below :

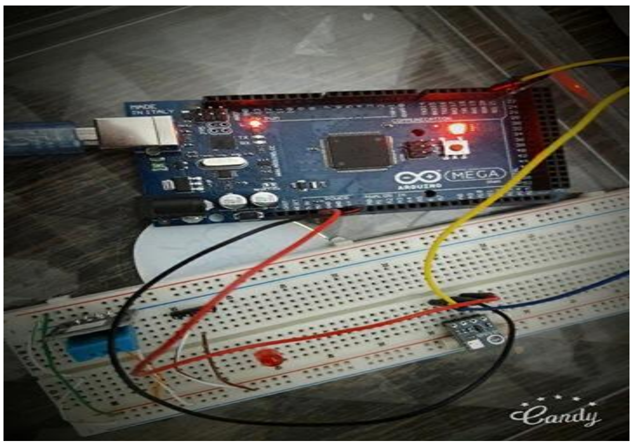


Figure 4.4: Arduino with BMP180

4.1.3 Interfacing Circuit between Arduino and SIM900 GSM/GPRS Shield

This shield is used for send all data to the server.

The SIM900 GSM/GPRS Shield has been connected to the Arduino as shown in the following figures :

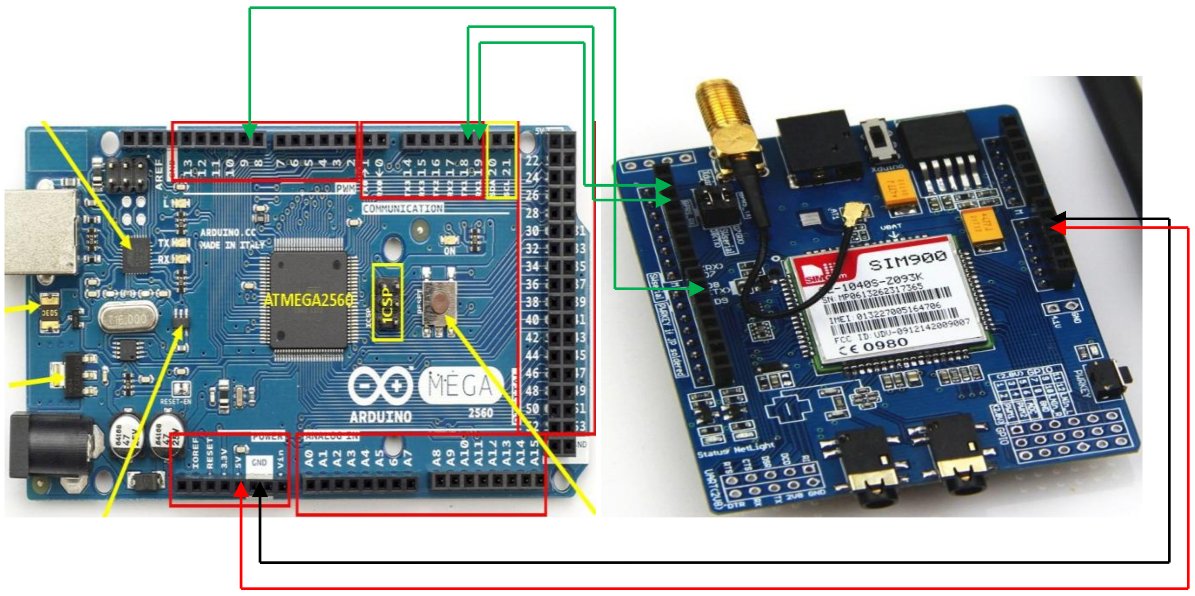


Figure 4.5: The interfacing circuit used to connect Arduino with SIM900

The connection in reality shown below :

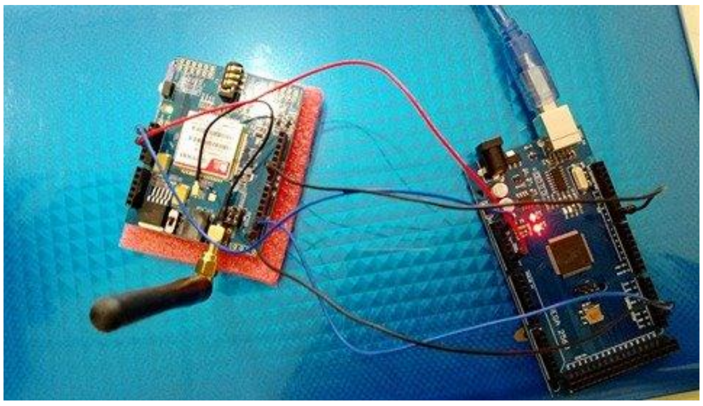


Figure 4.6: Arduino with SIM900 GSM/GPRS Shield

4.2 Interfacing codes

4.2.1 Code of interfacing between Arduinu and DHT11

```
#include <dht.h> //the library of the DHT11 that must downloaded by the user.

dht DHT; //declare a public variable that contained in dht11 library.

#define DHT11_PIN 3 // DHT11 signal pin

void setup(){

  Serial.begin(19600);

}

void loop()

{

  int chk=DHT.read11(DHT11_PIN); //get the reading from DHT11 pin

 Serial.print("TEM="); //print on serial monitor ‘’TEM=’’

 Serial.println(DHT.temperature); //print in new line the value of temperature

 Serial.print("hum="); //print on serial monitor ‘’hum=’’

  Serial.println(DHT.humidity); //print in new line the value of humidity

  delay(1000); //wait one second before get another reading

}
```

4.2.2 Code of interfacing between Arduinu and BMP180

```
#include <SFE_BMP180.h> // Your sketch must #include this library, and the Wire library.
#include <Wire.h> // (Wire is a standard library included with Arduino.):

SFE_BMP180 pressure; // You will need to create an SFE_BMP180 object, here called "pressure":


#define ALTITUDE 990.0 // Altitude of SparkFun's HQ in Boulder, CO. in meters
void setup()
{
  Serial.begin(9600);
  if (pressure.begin())//check if the sensor initialized or not
    Serial.println("BMP180 init success");//print “BMP180 init success” if the initialization success.
  Else
  {
    Serial.println("BMP180 init fail\n\n");//if the initialization not success print “BMP180 init fail”
    while(1); // Pause forever.
  }
}
void loop()
{
  char status;//declare a variable
  double T,P,p0,a;//declare variable
   status = pressure.startTemperature();
  if (status != 0)
  {
    delay(status); // Wait for the measurement to complete:
Serial.print(T);
    // Retrieve the completed temperature measurement. Note that the measurement is stored in the variable
T.Function returns 1 if successful, 0 if failure.
   status = pressure.getTemperature(T);
    if (status != 0)
    {
      Serial.print("temperature: ");// Print out the measurement:
      Serial.print(T,2);
       status = pressure.startPressure(3);
      if (status != 0)
       {
delay(status); // Wait for the measurement to complete:
// Retrieve the completed pressure measurement, Note that the measurement is stored in the variable P. Note
also that the function requires the previous temperature measurement (T), (If temperature is stable, you can do
one temperature measurement for a number of pressure measurements.)
function returns 1 if successful, 0 if failure.
status = pressure.getPressure(P,T);
        if (status != 0)
        {
          Serial.print("absolute pressure: ");          // Print out the measurement:
         Serial.print(P,2);
          Serial.print(" mb, ");
        } else Serial.println("error retrieving pressure measurement\n");
      } else Serial.println("error starting pressure measurement\n");
    }else Serial.println("error retrieving temperature measurement\n");
  } else Serial.println("error starting temperature measurement\n"); delay(5000);  // Pause for 5 seconds.
```

4.3 More Details About Web server

4.3.1 Introduction

There are a few basic terms need to understand at the beginning such web page, site ,and sever, since these expressions are heard all the time. It is easy to confuse these terms sometimes, since they refer to related but different functionalities. In fact, sometimes these terms misused in news reports and elsewhere, so getting them mixed up is understandable!

1. **web page**

A web page is a simple document displayable by a browser. Such document is written in the HTML language. A web page can embed a variety of different types of resources such as:

a. style information — controlling a page's look-and-feel
b. scripts — which add interactivity to the page
c. media — images, sounds, and videos.

2. **A website**

A website is a collection of linked web pages (plus their associated resources) that share a unique domain name. Each web page of a given website provides explicit links—most of the time in the form of clickable portion of text—that allow the user to move from one page of the website to another. To access a website, type its domain name in your browser address bar, and the browser will display the website's main web page, or homepage (casually referred as "the home"):

3. **Web server**

A web server is a computer hosting one or more websites. "Hosting" means that all the web pages and their supporting files are available on that computer. The web server will send any web page from the website it is hosting to any user's browser, per user request.

4. **Database**

A database is an organized collection of data. The data is typically organized to model aspects of reality in a way that supports processes requiring information. For example, modeling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

4.3.2 Save data to database

While broadcasting data over a dedicated web page ,it is useful to store data from connected sensors. This way enable to monitor live data, and to get historic information. It also allows to capture data from multiple data input devices and display them at any time and any way. Even though this could also be done with a dedicated web page by adding a little more code to your Arduino, it is easier to store it to a database and create a web page (or user interface) that reads data from the database.

4.3.2.1 SQL Database

SQL (pronounced "ess-que-el") stands for Structured Query Language. SQL is used to communicate with a database. According to ANSI (American National Standards Institute), t is the standard language for relational database management systems. SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database. Some common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, Access, Ingres, etc. Although most database systems use SQL, most of them also have their own additional proprietary extensions that are usually only used on their system. However, the standard SQL commands such as "Select", "Insert", "Update", "Delete", "Create", and "Drop" can be used to accomplish almost everything that one needs to do with a database.

A relational database system contains one or more objects called tables. The data or information for the database are stored in these tables. Tables are uniquely identified by their names and are comprised of columns and rows. Columns contain the column name, data type, and any other attributes for the column. Rows contain the records or data for the columns.

4.3.2.2 Creation of SQL database

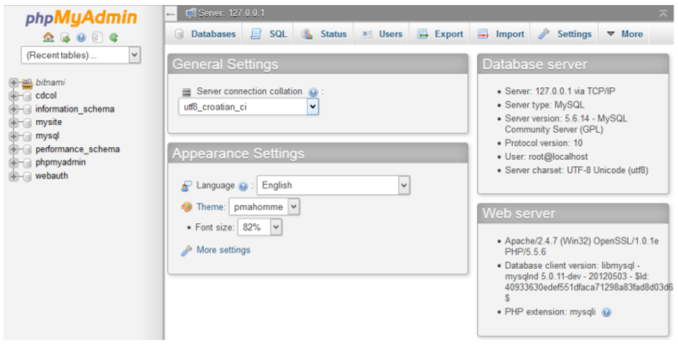
To create a web sql ,one thing should be available :

1. Online website hosting: a web hosting will be need to run php and mysql database(like cpanel).
2. Local server environment like XAMP.

The main difference between them that the first one has a real ip address

4.3.2.3 Prepare the database

If web database service running is available or XAMPP installed just log into phpMyAdmin interface to create a new database.


phpMyAdmin

(Recent tables) ...

- bitnami
- cdcol
- information_schema
- mysite
- mysql
- performance_schema
- phpmyadmin
- webauth

Server: 127.0.0.1

Databases | SQL | Status | Users | Export | Import | Settings | More

General Settings

Server connection collation: utf8_croatian_ci

Appearance Settings

Language: English

Theme: pmahomme

Font size: 82%

More settings

Database server

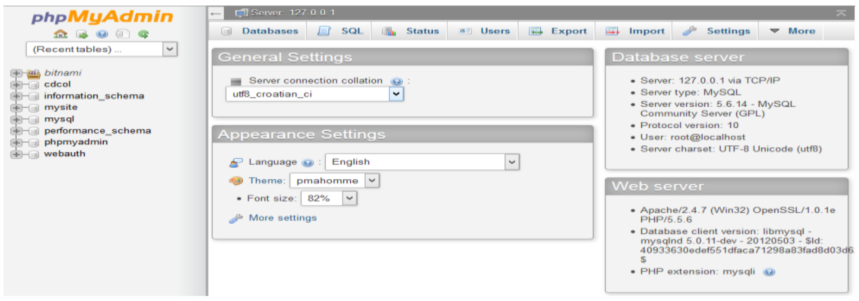
- Server: 127.0.0.1 via TCP/IP
- Server type: MySQL
- Server version: 5.6.14 - MySQL Community Server (GPL)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

Web server

- Apache/2.4.7 (Win32) OpenSSL/1.0.1e PHP/5.5.6
- Database client version: libmysql - mysqlnd 5.0.11-dev - 20120503 - $Id: 40933630edef551dfaca71298a83fad8d03d6 $
- PHP extension: mysqli

Figure 4.7: phpMyAdmin interface

Under the tab “Databases” create a new database. Pick a name and collation (depending on your keyboard localization, or you can leave it as is) and press “Create”.


phpMyAdmin

(Recent tables) ...

- bitnami
- cdcol
- information_schema
- mysite
- mysql
- performance_schema
- phpmyadmin
- webauth

Server: 127.0.0.1

Databases | SQL | Status | Users | Export | Import | Settings | More

General Settings

Server connection collation: utf8_croatian_ci

Appearance Settings

Language: English

Theme: pmahomme

Font size: 82%

More settings

Database server

- Server: 127.0.0.1 via TCP/IP
- Server type: MySQL
- Server version: 5.6.14 - MySQL Community Server (GPL)
- Protocol version: 10
- User: root@localhost
- Server charset: UTF-8 Unicode (utf8)

Web server

- Apache/2.4.7 (Win32) OpenSSL/1.0.1e PHP/5.5.6
- Database client version: libmysql - mysqlnd 5.0.11-dev - 20120503 - $Id: 40933630edef551dfaca71298a83fad8d03d6 $
- PHP extension: mysqli

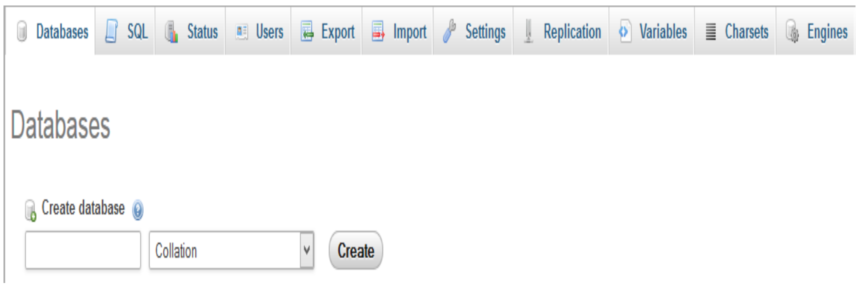
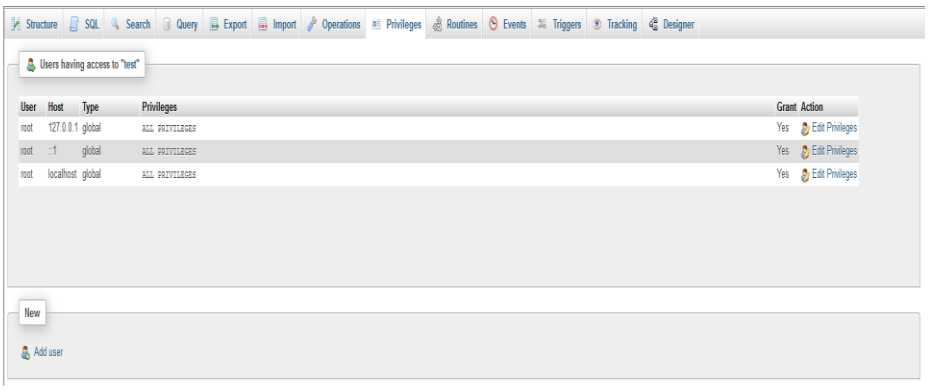


Figure 4.8: How to create a new database

When you create the database select the tab “Privileges”, from the tab menu.

Next need to create a user that will have access to the database. Press "Add user" and in the new menu enter a username and a password for the user that we are going to use for database access. other fields could be blank, but the checkbox Grant all privileges on database



Users having access to "test"

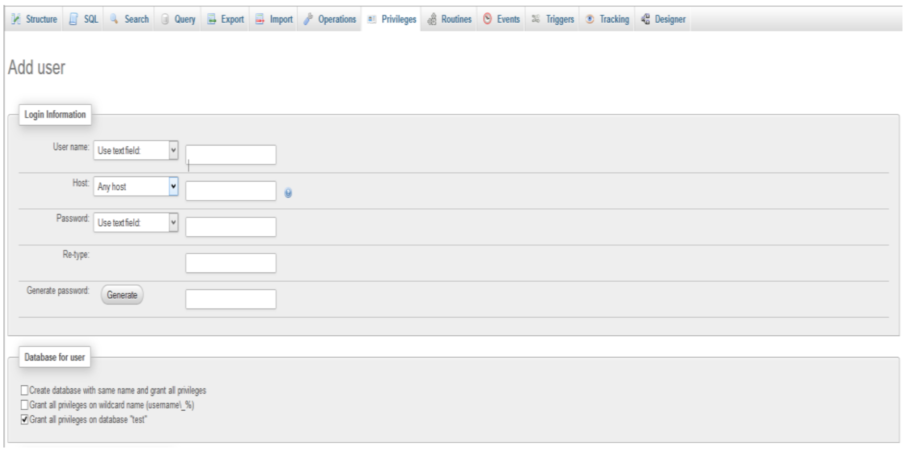
User	Host	Type	Privileges	Grant	Action
root	127.0.0.1	global	ALL PRIVILEGES	Yes	Edit Privileges
root	::1	global	ALL PRIVILEGES	Yes	Edit Privileges
root	localhost	global	ALL PRIVILEGES	Yes	Edit Privileges

New

Add user

Figure 4.9: How to create a user that will have access to the database

We need a new table that we are going to write data to. Choose "SQL" from the tab menu:



Add user

Login Information

User name:	Use text field	
Host:	Any host	
Password:	Use text field	
Re-type:		
Generate password:	Generate	

Database for user

☐ Create database with same name and grant all privileges
☐ Grant all privileges on wildcard name (username_%)
☑ Grant all privileges on database "test"

Put in this code and press "Go".

CREATE TABLE 'test'.'sensor' ('id' INT NOT NULL AUTO_INCREMENT PRIMARY KEY, 'time' TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP, 'value' VARCHAR(10) NOT NULL)

4.3.2.4 Create files that will capture data sent from Arduino and write it to database

The easiest way to get data from Arduino to your database is to use php and HTTP GET request method.

GET – Requests a representation of the specified resource. Requests using GET should only retrieve data and should have no other effect.

Basically, GET is used for sending limited amount of data to a webpage, ie. a GET request looks like this:

```
http://www.yourwebpage.com/write_data.php?data1=value1;data2=value2;data3=value3...
```

Still this doesn't write data directly into the database, so php file(for example, the "write_data.php" file) should be made to receive data from Arduino which will write data to database.

4.4 conceptual design of the server

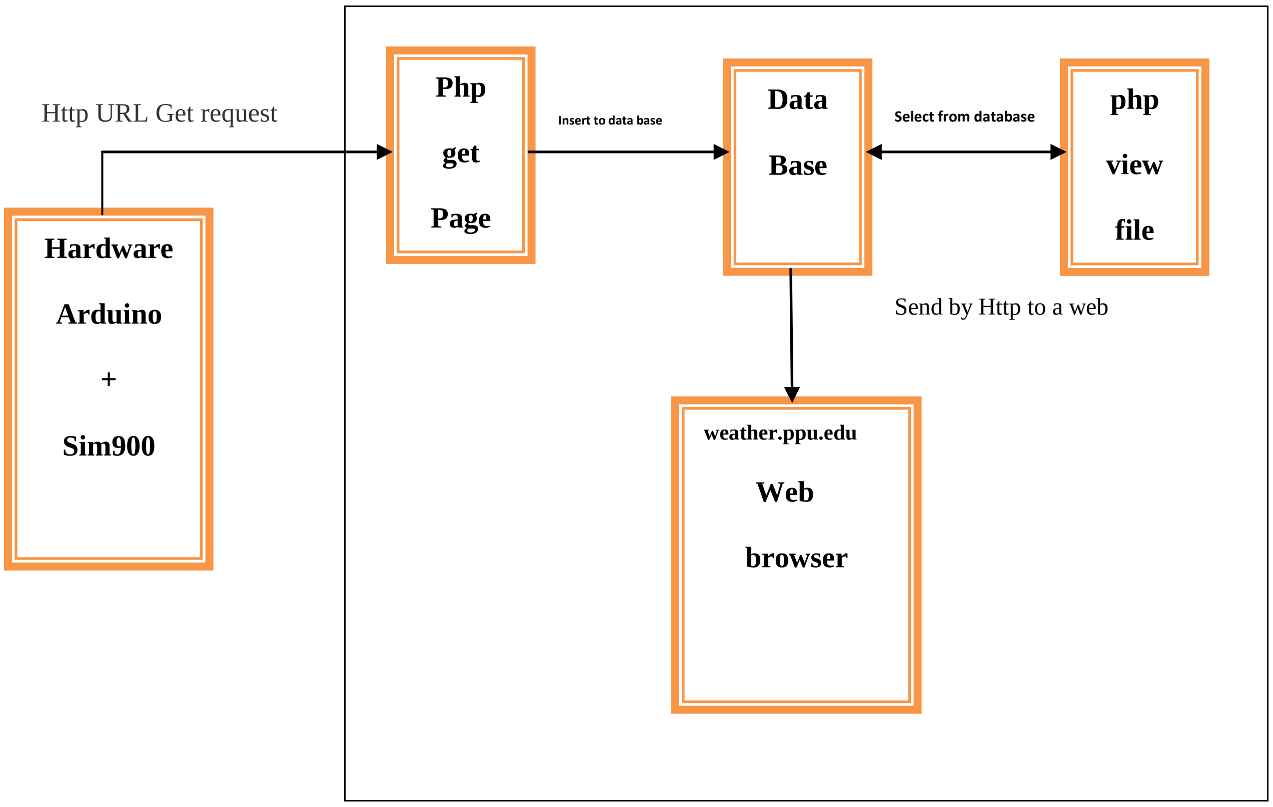


Figure 4.10: conceptual design of the server

4.4.1 The PHP application consists of 4 main files:

conn_db.php: this file is loaded every time we need to access to the database. It's loaded in the beginning of the almost each file. It contains a function that returns a new connection to be used by the PHP to execute query's to the DB. You need to store the **DB configurations (hostname, database, user, password)** in this file.

```
<?php
include "config.php";
// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);
// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
} ?>
```

4.4.2 receive_data_GET.php

when the Arduino sends GET requests to the server, The PHP receives the values sent in the request and executes an insertion query with those values.

```
<?php

$location = $_GET['location'];

$temperature = $_GET['temperature'];

$humidity = $_GET['humidity'];

$pressure = $_GET['pressure'];

$password = $_GET['password'];

/*$all_date =date("y-m-d h:i:s");

$format_date = new Datetime($all_date);

$date_day=$format_date->format('Y-m-d');

$hour=$format_date->format('h');

$minute=$format_date->format('i');*/

if($password=="123123"){

include "conn_db.php";

$sql="INSERTINTOparometers(location,temperature,humidity,pressure)VALUES
('".$location."','".$temperature."','".$humidity."','".$pressure."')";

if ($conn->query($sql) === TRUE

{echo "New record created successfully";} else {echo "Error: " . $sql . "<br>" .
$conn->error;}

$conn->close();

}else{echo "error password";}

//http://localhost/js-html/ppu-
weather/receive_data_GET.php?location=location1&password=123123&temperat
ure=32&humidity=10&pressure=1000

?>
```

4.4.3 index.php

this is the website landing page. It displays the values that are stored in the database on a web browser

```
<?php

// Start the session

session_start(); ?>

<?php include 'header.php'; ?>

<?php include 'function.php';?>

<form action="index.php" method="post">

<div class="component">

<br></br>

<div style="width:89%; display:inline-block;">

<span>

<?php $date_top =date("y-m-d h:i:s");

$datea_top = new Datetime($date_top);

echo $datea_top->format('D d-m-Y')?>

</span>

</div>

<divstyle="width:10%; display:inline-block;">

  <span id='hrs'>0</span>:<span id='mins'>0</span>:<span id='secs'>0</span>

</div> <br></br>

 <select name="location" class="button1" onchange="this.form.submit()" >

              <option value="location1" <?php if ($_POST["location"]=="location1") echo
"selected";?> >Building B</option>

              <option value="location2" <?php if ($_POST["location"]=="location2") echo
"selected";?> >Building C</option>

    </select>
```

```
<?php

 (isset($_POST['location']))

$location = $_POST['location'];

 }else{$location="location1";?>

<div class="sticky-wrap">

<table class="table2" style="margin: auto; width: 80%;">

<caption>Current Measurments</caption><tr>

<th>Temperature (C̊  &nbsp; )</th>

<th>Temp Max (C̊  &nbsp; )</th>

<th>Temp Min (C̊  &nbsp; )</th>

<th>Humidity (%)</th>

<th>Pressure (hpa)</th>

</tr><tr><td><?php current_temp($location); ?></td>

<td><?php temp_max($location); ?></td>

<td><?php temp_min($location); ?></td>

<td><?php current_hum($location); ?></td>

<td><?php current_pre($location); ?></td></tr>

</table> <table class="table2" style="margin:auto; width: 80%;"<caption>Average In the
last 3 Hours</caption><tr>

<th>Temperature (C̊   &nbsp; )</th><th>Humidity (%)</th><th>Pressure (hpa)</th></tr>

<tr><td><?php cal_average_temp($location); ?></td><td><?php
cal_average_hum($location); ?></td><td><?php cal_average_pre($location);
?></td></tr></table></div><br></br>

<select name="data_charting" class="button1" onchange="this.form.submit()"><option
value="temperature" <?php if ($_POST["data_charting"]=="temperature") echo
"selected";?>>Temperature</option>
```

```
<option value="humidity" <?php if ($_POST["data_charting"]=="humidity") echo "selected";?>>Humidity</option>

<option value="pressure" <?php if ($_POST["data_charting"]=="pressure") echo "selected";?> >Pressure</option> </select>

<?php

if(isset($_POST['data_charting'])) $data_charting = $_POST['data_charting'];

}else{$data_charting="temperature";}?>

<select name="interval" class="button1" onchange="this.form.submit()">

<option value="day" <?php if ($_POST["interval"]=="day") echo "selected";?> >Last Day</option>

<option value="week" <?php if ($_POST["interval"]=="week") echo "selected";?>>Last Week</option>

<option value="month" <?php if ($_POST["interval"]=="month") echo "selected";?>>Last Month</option>

<option value="year" <?php if ($_POST["interval"]=="year") echo "selected";?>>Last Year</option>

</select<?php if(isset($_POST['interval'])){

$interval = $_POST['interval'];}

else{$interval="day";?> <!—

<input type="submit" class="button2" name="submit_history" value="  History  "/

<?php if ($_SERVER['REQUEST_METHOD'] === 'POST') {

if (isset($_POST['submit_history'])) {$fun_type="history"} }else {$fun_type="history";}?>-->

<div id="chartContainer_one" style="height: 250px; width: 80%;margin: auto;border-radius: 18px;"></div>

">
```

```
<?php

$locations_dic=array("location1"=>"Building B","location2"=>"Building C");

$interval_dic=array("day"=>"LastDay","week"=>"LastWeek","month"=>"LastMonth","year"=>
"Last Year");?><script type="text/javascript">

window.onload = function() {


        //**************************************************************************
*************

var charttemp = new CanvasJS.Chart("chartContainer_one", {theme: "theme4",zoomEnabled:
true,title: {

text:<?php echo "\"".$locations_dic[$location]."-".$data_charting." -
".$interval_dic[$interval]."\"" ?>},animationEnabled: true,animationDuration: 2000,data: [{

type: "line" ,/* "column" "area" "bar" "spline" "pie" "line" */

        dataPoints:<?php generate_dataPoints($location,$data_charting,$interval);
?>}]});charttemp.render();}</script></form>

<?php include "footer.php"; ?>
```

4.4.4 Function.php

This file performs any operation on the stored data, such as determine the average or maximum or minimum of a given data within a cretin period .

```
<?php

function generate_dataPoints($location,$data_showing,$interval){

//$data_showing==> temperature - humidity - pressure

//$interval ==> day , week, year

include "conn_db.php";

$sql = "SELECT * FROM parometers where location='".$location."' and
current_time_stamp >= (CURDATE() - INTERVAL 1 ".$interval.")";$result = $conn-
>query($sql); $data_points="[";

 if ($result->num_rows > 0) {

while($row = $result->fetch_assoc()) {

$data_points=$data_points."{y:".$row[$data_showing].",label:
\"".$row["current_time_stamp"]."\"},";}}

$data_points=$data_points."]";

$conn->close();

echo $data_points;

/*

$data_points="[{ y: 12, label: \"10:00 am\" },

                    { y: 31, label: \"11:00 am\" },

                    { y: 3, label: \"12:00 pm\" },

                    { y: 46, label: \"01:00 pm\" },

                    { y: 30, label: \"02:00 pm\" },

                    { y: 14, label: \"03:00 pm\" },

                    { y: 24, label: \"04:00 pm\" },

                    { y: 45, label: \"05:00 pm\" },

                    { y: 20, label: \"06:00 pm\" },

                    { y: 45, label: \"07:00 pm\" },

                    { y: 45, label: \"08:00 pm\" },

                    { y: 45, label: \"09:00 pm\" },

                    { y: 45, label: \"10:00 pm\" },

                    { y: 45, label: \"11:00 pm\" },

                    { y: 20, label: \"12:00 am\" },
{ y: 45, label: \"01:00 am\" },
                    { y: 33, label: \"02:00 am\" },
                    { y: 32, label: \"03:00 am\" },
                    { y: 23, label: \"04:00 am\" },
                    { y: 23, label: \"05:00 am\" },
                    { y: 23, label: \"06:00 am\" },
                    { y: 15, label: \"07:00 am\" },
                    { y: 45, label: \"08:00 am\" },
                    { y: 45, label: \"09:00 am\" },]";
*/


}function cal_average_temp($location){
include "conn_db.php";
$sql = "SELECT * FROM parometers where location='".$location."' and
current_time_stamp >= (CURDATE() - INTERVAL 3 HOUR)";
          $result = $conn->query($sql);
          $average=0;
          $total=0;
          $count=0;
          if ($result->num_rows > 0)
          {
          while($row = $result->fetch_assoc()) {
          $total+=(int)$row["temperature"];
$count+=1;}$average=$total/$count;}echo (int)$average;
$conn->close();}
function cal_average_hum($location){
include "conn_db.php";
$sql = "SELECT * FROM parometers where location='".$location."' and
current_time_stamp >= (CURDATE() - INTERVAL 3 HOUR)";
```

```
            $result = $conn->query($sql);

            $average=0;

            $total=0;

            $count=0;

            if ($result->num_rows > 0)

            {while($row = $result->fetch_assoc()) {

            $total+=(int)$row["humidity"];$count+=1;}

$average=$total/$count;

            }echo (int)$average;

            $conn->close();}

function cal_average_pre($location){

include "conn_db.php";

$sql = "SELECT * FROM parometers where location='".$location."' and
current_time_stamp >= (CURDATE() - INTERVAL 3 HOUR)";

$result = $conn->query($sql);

            $average=0;

            $total=0;

            $count=0;

            if ($result->num_rows > 0)

            {while($row = $result->fetch_assoc()) {

            $total+=(int)$row["pressure"];

            $count+=1;}

$average=$total/$count;

            }

echo (int)$average;$conn->close();}function current_temp($location)

include "conn_db.php";

$sql = "SELECT * FROM parometers where location='".$location."' ORDER BY
current_time_stamp DESC LIMIT 1 ";

$result = $conn->query($sql);$temp="";
```

```
            if ($result->num_rows > 0)

            {                        $row = $result->fetch_assoc();        $temp=$row["temperature"];

            }

echo $temp;$conn->close();}

function current_hum($location){

//echo $location;include "conn_db.php";

            $sql = "SELECT * FROM parometers where location='".$location."' ORDER BY
current_time_stamp DESC LIMIT 1 ";

$result = $conn->query($sql);

$hum="";

if ($result->num_rows > 0) {$row = $result->fetch_assoc();$hum=$row["humidity"];}

echo $hum;$conn->close();}

function current_pre($location){

include "conn_db.php";

$sql = "SELECT * FROM parometers where location='".$location."' ORDER BY
current_time_stamp DESC LIMIT 1 ";

            $result = $conn->query($sql);

            $pre="";

            if ($result->num_rows > 0)

            {

            $row = $result->fetch_assoc();

                        $pre=$row["pressure"];}echo $pre;

            $conn->close();}

function temp_max($location){

include "conn_db.php";

$sql = "SELECT * FROM parometers where location='".$location."' and
current_time_stamp >= (CURDATE() - INTERVAL 1 DAY)";

$result = $conn->query($sql);

$max_temp=-50;

            if ($result->num_rows > 0)
```

```
{
            while($row = $result->fetch_assoc())
            {
            if((int)$row["temperature"]>$max_temp)$max_temp=(int)$row["temperature"];}}$
conn->close();
echo $max_temp;
}function temp_min($location){
            include "conn_db.php";
$sql = "SELECT * FROM parometers where location='".$location."' and
current_time_stamp >= (CURDATE() - INTERVAL 1 DAY)";
            $result = $conn->query($sql);
            $min_temp=100;
            if ($result->num_rows > 0)
            {while($row = $result->fetch_assoc())
                        {
            if((int)$row["temperature"]<$min_temp)$min_temp=(int)$row["temperature"];}
            }
            $conn->close();
echo $min_temp;
}
```

4.5 What is HTTP?

The Hypertext Transfer Protocol (HTTP) is designed to enable communications between clients and servers. HTTP works as a request-response protocol between a client and server.

A web browser may be the client, and an application on a computer that hosts a web site may be the server.

Example: A client (browser) submits an HTTP request to the server; then the server returns a response to the client. The response contains status information about the request and may also contain the requested content.

There are Two HTTP Request Methods: GET and POST

Two commonly used methods for a request-response between a client and server are: GET and POST.

1. GET - Requests data from a specified resource
2. POST - Submits data to be processed to a specified resource

	GET	**POST**
Bookmarked	Can be bookmarked	Cannot be bookmarked
BACK button/Reload	Harmless	Data will be re-submitted (the browser should alert the user that the data are about to be re-submitted)
Cached	Can be cached	Not cached
Encoding type	application/x-www-form-urlencoded	application/x-www-form-+urlencoded or multipart/form-data. Use multipart encoding for binary data
History	Parameters remain in browser history	Parameters are not saved in browser history
Restrictions on data length	Yes, when sending data, the GET method adds the data to the URL; and the length of a URL is limited (maximum URL length is 2048 characters)	No restrictions

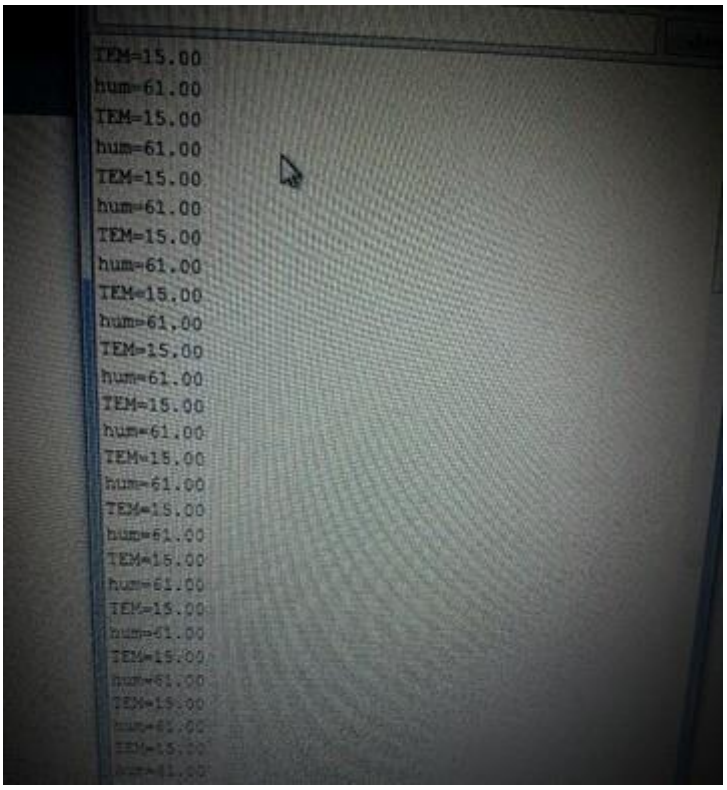
Restrictions on data type	Only ASCII characters allowed	No restrictions. Binary data is also allowed
Security	GET is less secure compared to POST because data sent is part of the URL Never use GET when sending passwords or other sensitive information!	POST is a little safer than GET because the parameters are not stored in browser history or in web server logs
Visibility	data is visible to every one in the URL	Data is not displayed in the URL

CHAPTER FIVE

RESULT AND PERFORMANCE

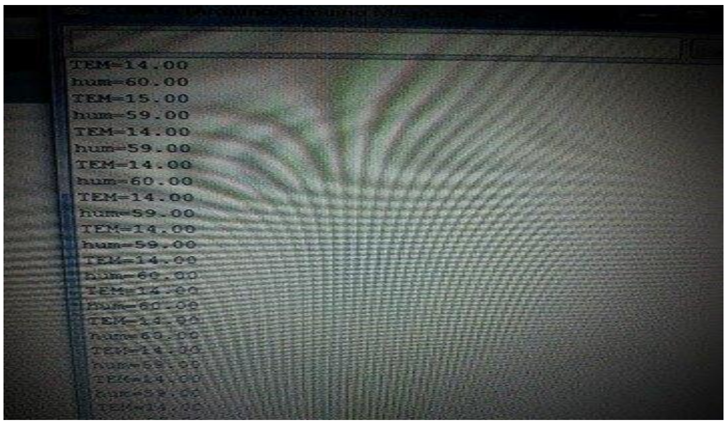
The group of this project made the component testing under different enviroments to make sure from its validity; these testings are take a place in this chapter.

5.1 DHT11 Testing



```
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
TEM=15.00
hum=61.00
```

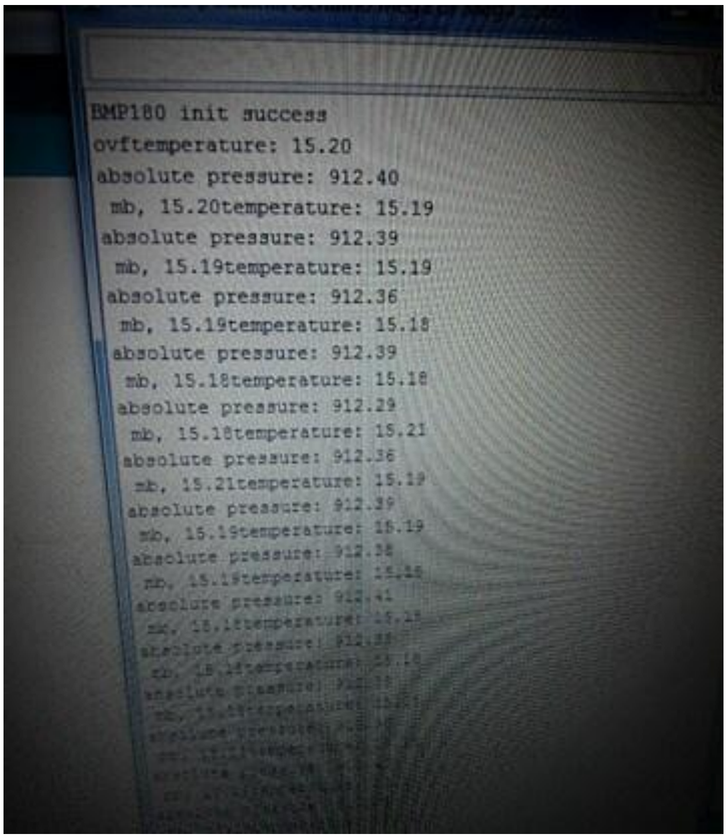
Figure 5.1: DHT11 Testing at room environment



```
TEM=14.00
hum=60.00
TEM=15.00
hum=59.00
TEM=14.00
hum=59.00
TEM=14.00
hum=60.00
TEM=14.00
hum=59.00
TEM=14.00
hum=59.00
TEM=14.00
hum=60.00
TEM=14.00
hum=60.00
TEM=14.00
hum=60.00
TEM=14.00
hum=59.00
TEM=14.00
hum=59.00
TEM=14.00
```

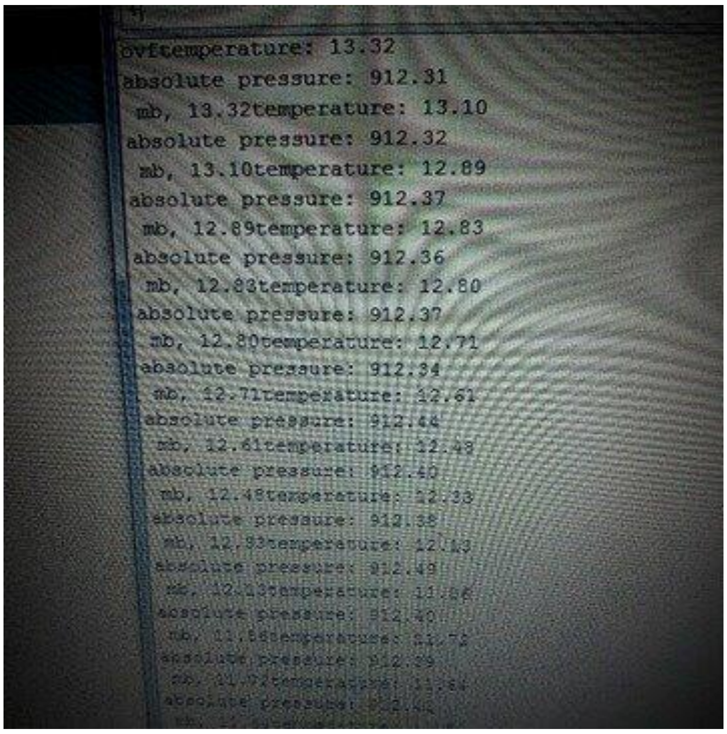
Figure 5.2: DHT11 Testing outside

5.2 BMP180 Testing



```
BMP180 init success
ovftemperature: 15.20
absolute pressure: 912.40
 mb, 15.20temperature: 15.19
absolute pressure: 912.39
 mb, 15.19temperature: 15.19
absolute pressure: 912.36
 mb, 15.19temperature: 15.18
absolute pressure: 912.39
 mb, 15.18temperature: 15.18
absolute pressure: 912.29
 mb, 15.18temperature: 15.21
absolute pressure: 912.36
 mb, 15.21temperature: 15.19
absolute pressure: 912.39
 mb, 15.19temperature: 15.19
absolute pressure: 912.38
 mb, 15.19temperature: 15.16
absolute pressure: 912.41
 mb, 15.16temperature: [illegible]
absolute pressure: [illegible]
[illegible]
```

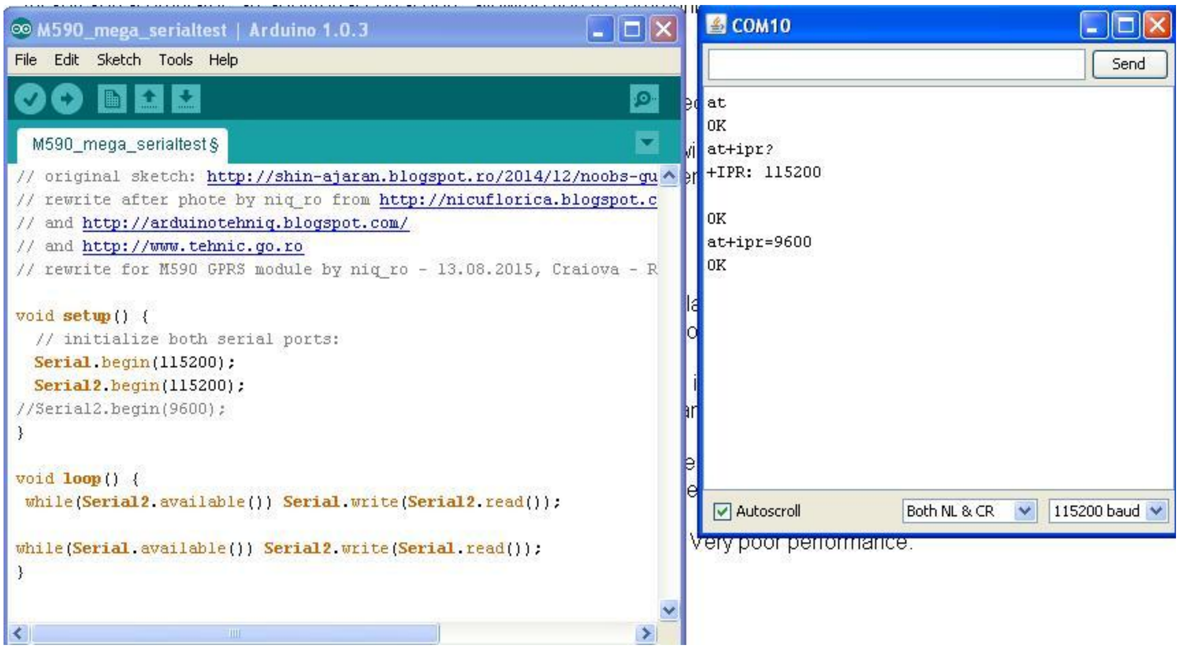
Figure 5.3: BMP testing inside



```
ovftemperature: 13.32
absolute pressure: 912.31
 mb, 13.32temperature: 13.10
absolute pressure: 912.32
 mb, 13.10temperature: 12.89
absolute pressure: 912.37
 mb, 12.89temperature: 12.83
absolute pressure: 912.36
 mb, 12.83temperature: 12.80
absolute pressure: 912.37
 mb, 12.80temperature: 12.71
absolute pressure: 912.34
 mb, 12.71temperature: 12.61
absolute pressure: 912.44
 mb, 12.61temperature: 12.48
absolute pressure: 912.40
 mb, 12.48temperature: 12.33
absolute pressure: 912.38
 mb, 12.33temperature: 12.13
absolute pressure: 912.49
 mb, 12.13temperature: 11.86
absolute pressure: 912.40
 mb, 11.86temperature: 11.72
absolute pressure: 912.39
 mb, 11.72temperature: 11.64
absolute pressure: [illegible]
[illegible]
```

Figure 5.4: BMP180 testing outside

5.3 SIM900 GSM/GPRS Shield Testing



```
// original sketch: http://shin-ajaran.blogspot.ro/2014/12/noobs-gu
// rewrite after phote by niq_ro from http://nicuflorica.blogspot.c
// and http://arduinotehniq.blogspot.com/
// and http://www.tehnic.go.ro
// rewrite for M590 GPRS module by niq_ro - 13.08.2015, Craiova - R

void setup() {
  // initialize both serial ports:
  Serial.begin(115200);
  Serial2.begin(115200);
//Serial2.begin(9600);
}

void loop() {
 while(Serial2.available()) Serial.write(Serial2.read());

while(Serial.available()) Serial2.write(Serial.read());
}
```

```
at
OK
at+ipr?
+IPR: 115200

OK
at+ipr=9600
OK
```

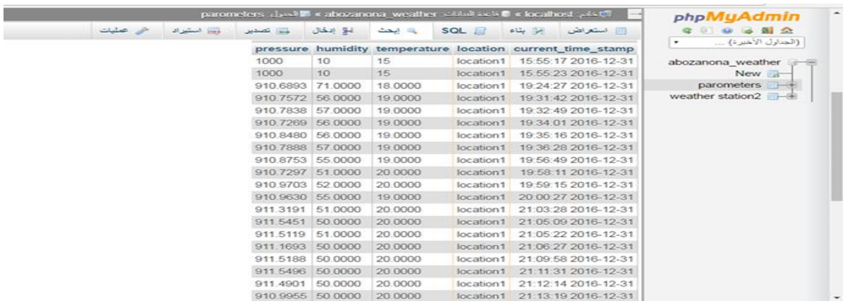
Figure 5.5: SIM900 GSM/GPRS Shield Testing

5.4 Results of the system

After assembling all components and codes the group reached to their requirement that this project made for.

By putting the code of hole system-that shown in the appendix D- and operate it, the group reach to many results that are displayed in the following pictures .

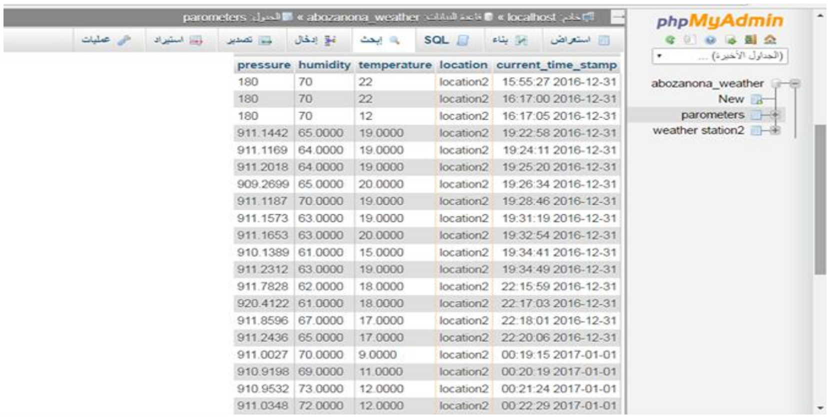
The first one show the readings that reaches to the database of the server from the first location.



pressure	humidity	temperature	location	current_time_stamp
1000	10	15	location1	15:55:17 2016-12-31
1000	10	15	location1	15:55:23 2016-12-31
910.6893	71.0000	18.0000	location1	19:24:27 2016-12-31
910.7572	56.0000	19.0000	location1	19:31:42 2016-12-31
910.7838	57.0000	19.0000	location1	19:32:49 2016-12-31
910.7269	56.0000	19.0000	location1	19:34:01 2016-12-31
910.8480	56.0000	19.0000	location1	19:35:16 2016-12-31
910.7888	57.0000	19.0000	location1	19:36:28 2016-12-31
910.8753	55.0000	19.0000	location1	19:56:49 2016-12-31
910.7297	51.0000	20.0000	location1	19:58:11 2016-12-31
910.9703	52.0000	20.0000	location1	19:59:15 2016-12-31
910.9630	55.0000	19.0000	location1	20:00:27 2016-12-31
911.3191	51.0000	20.0000	location1	21:03:28 2016-12-31
911.5451	50.0000	20.0000	location1	21:05:09 2016-12-31
911.5119	51.0000	20.0000	location1	21:05:22 2016-12-31
911.1693	50.0000	20.0000	location1	21:06:27 2016-12-31
911.5188	50.0000	20.0000	location1	21:09:58 2016-12-31
911.5496	50.0000	20.0000	location1	21:11:31 2016-12-31
911.4901	50.0000	20.0000	location1	21:12:14 2016-12-31
910.9955	50.0000	20.0000	location1	21:13:19 2016-12-31

Figure 5.6: Database of the first location

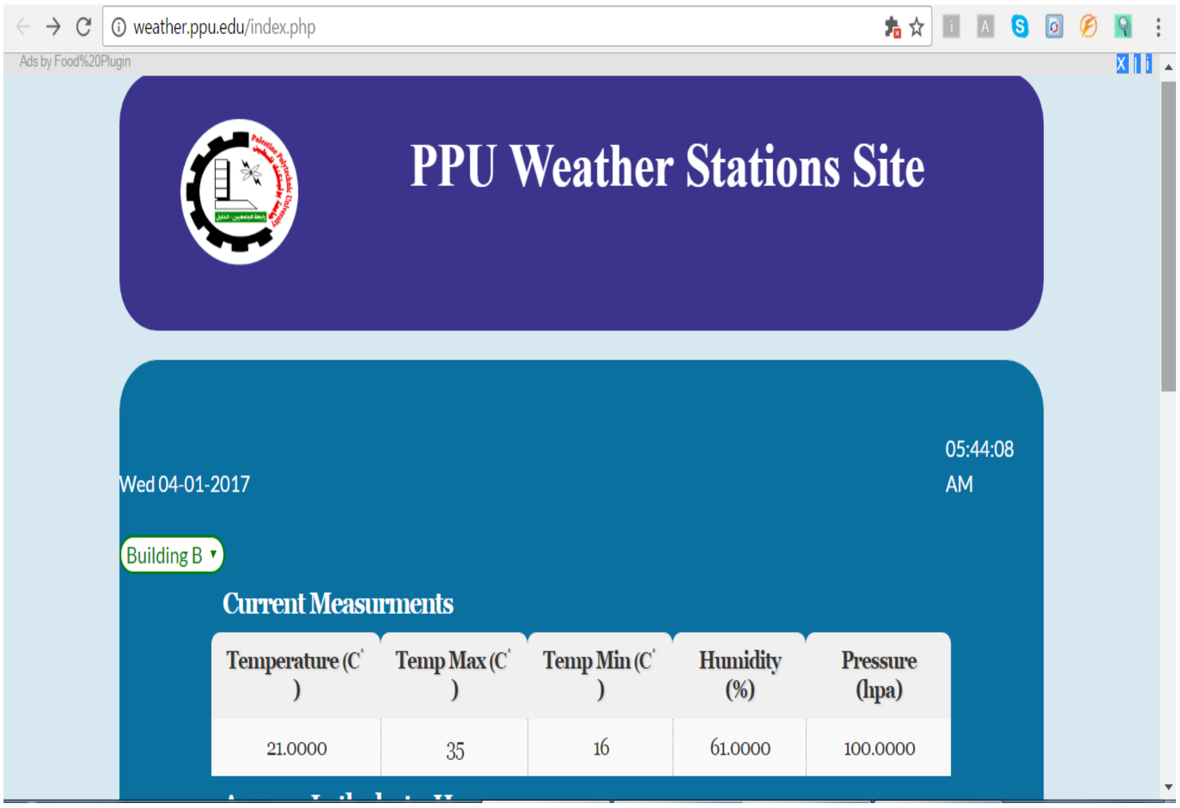
The second shows the readings that reach to the database of the server from the second location.



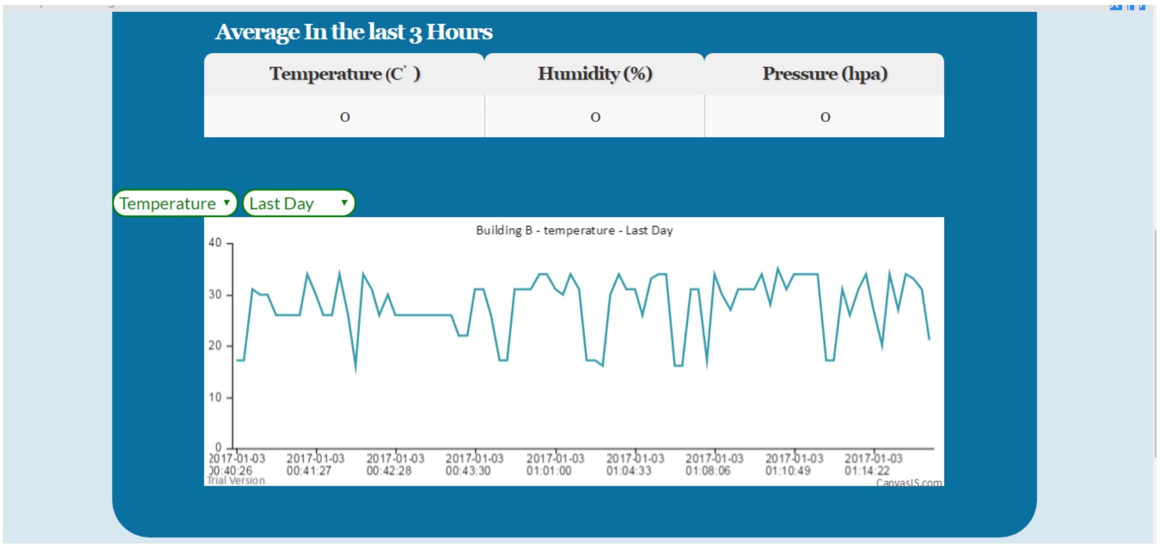
pressure	humidity	temperature	location	current_time_stamp
180	70	22	location2	15:55:27 2016-12-31
180	70	22	location2	16:17:00 2016-12-31
180	70	12	location2	16:17:05 2016-12-31
911.1442	65.0000	19.0000	location2	19:22:58 2016-12-31
911.1169	64.0000	19.0000	location2	19:24:11 2016-12-31
911.2018	64.0000	19.0000	location2	19:25:20 2016-12-31
909.2699	65.0000	20.0000	location2	19:26:34 2016-12-31
911.1187	70.0000	19.0000	location2	19:28:46 2016-12-31
911.1573	63.0000	19.0000	location2	19:31:19 2016-12-31
911.1653	63.0000	20.0000	location2	19:32:54 2016-12-31
910.1389	61.0000	15.0000	location2	19:34:41 2016-12-31
911.2312	63.0000	19.0000	location2	19:34:49 2016-12-31
911.7828	62.0000	18.0000	location2	22:15:59 2016-12-31
920.4122	61.0000	18.0000	location2	22:17:03 2016-12-31
911.8596	67.0000	17.0000	location2	22:18:01 2016-12-31
911.2436	65.0000	17.0000	location2	22:20:06 2016-12-31
911.0027	70.0000	9.0000	location2	00:19:15 2017-01-01
910.9198	69.0000	11.0000	location2	00:20:19 2017-01-01
910.9532	73.0000	12.0000	location2	00:21:24 2017-01-01
911.0348	72.0000	12.0000	location2	00:22:29 2017-01-01

Figure 5.7: Database of the second location

The last two pictures show the site that the users reached to; to know the condition of the weather in any place they want without any effort.



Temperature (C°)	Temp Max (C°)	Temp Min (C°)	Humidity (%)	Pressure (hpa)
21.0000	35	16	61.0000	100.0000



Average In the last 3 Hours

Temperature (C˚)	Humidity (%)	Pressure (hpa)
0	0	0

Figure 5.8: The site of the project

CHAPTER SIX

Conclusion and Recommendations

6.1 Problems

Many problems, challenges, and issues have been raised during the work on the project. Many experiments, ideas and researches have been carried out to deal with different problems, some of these problems are :

1. The availability of more accurate and appropriate sensors to the project.
2. Problems while establishing GPRS communication; due to the limitation on the frequency band imposed on the local mobile networks that cause a low coverage in the GPRS service.

6.2 Acquired Learning Outcomes

1. We have learnt how to deal with Arduino as hardware and as software.
2. We have learnt some basics of database building.
3. We have learnt how to interface the Arduino with different type of sensors and with GPRS module.
4. We have learnt the ways of solving problems.

6.3 Recommendations

At the end some ideas can added to develop or extend this project; by remembering that this project is scalable and can contains a huge number of node:

1. Using solar cell for each node is a good idea.
2. Using more precise sensors as much as possible.

REFRENCES

[1] http://www.robotroom.com/Weather-Station.html

[2] http://www.environmental-expert.com/products/environdata-easidata-mark-4-183915

[3]A.Sicher, R.Heaton GPRS technology overview,Dell computer Corporation,2002

[4] Yokogawa Electric Corporation, SCADA –RTU Communication Using GPRS, http://www. Yokogawa.com.

[5] Internet Protocol – DARPA Internet Program Protocol Specification (September 1981)

[6] The University of New South Wales: GPRS Optimization, Telenor: GPRS FAQ, Networks and Telecommunications Research Group: GPRS, RTX: GSM/GPRS Technology, India Parenting: Why Enable GPRS on Your Mobile Phone.

[7] http://www.arduino.org/learning/getting-started/what-is-arduino

[8] http://www.ti.com/lit/ds/symlink/lm35.pdf

[9]http://www.dx.com/p/geeetech-bmp085-breakout-barometric-pressure-sensor-for-arduino-red-369740#.Vy0X_tKDGko

[10] Arduino Microcontroller Guide W. Durfee, University of Minnesota ver. oct-2011

[11] http://www.sqlcourse.com

[12] http://www.developershome.com/sms/atCommandsIntro.asp

[13] https://www.arduino.cc/en/Main/ArduinoBoardMega2560

Appendices

Appendix A

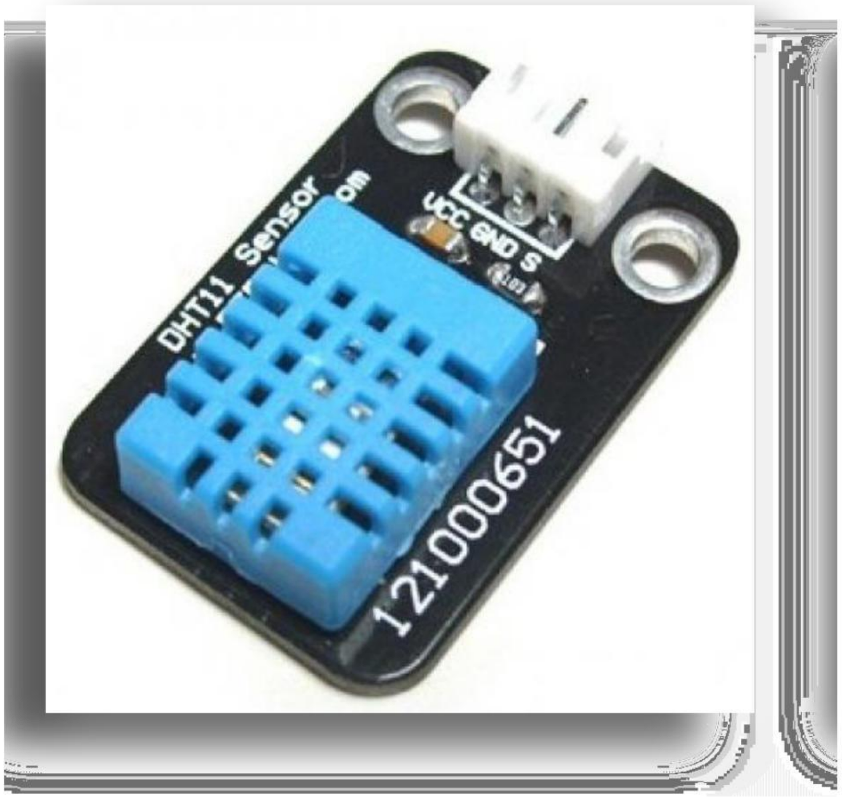
DHT11 Humidity &Temperature Sensor

D-Robotics UK (www.droboticsonline.com)

DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output.

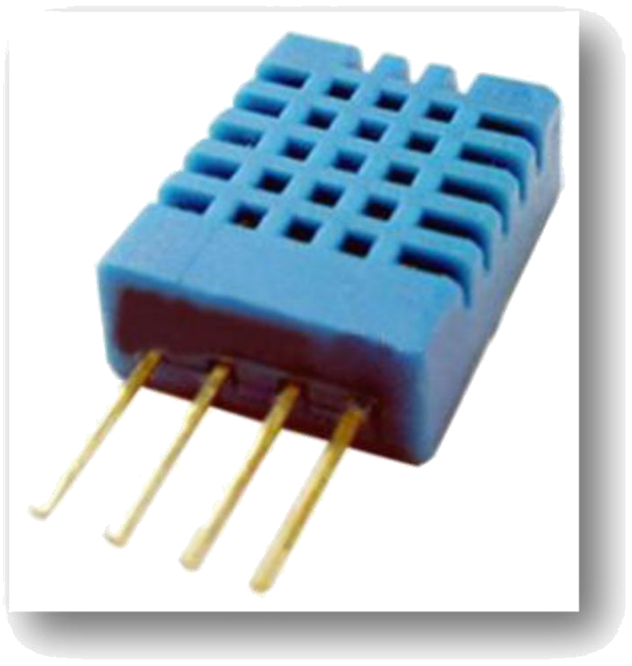
D-Robotics 7/30/2010

DHT 11 Humidity & Temperature
Sensor



1. Introduction

This DFRobot DHT11 Temperature & Humidity Sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.



Each DHT11 element is strictly calibrated in the laboratory that is extremely accurate on humidity calibration. The calibration coefficients are stored as programmes in the OTP memory, which are used by the sensor's internal signal detecting process. The single-wire serial interface makes system integration quick and easy. Its small size, low power consumption and up-to-20 meter signal transmission making it the best choice for various applications, including those most demanding ones. The component is 4-pin single row pin package. It is convenient to connect and special packages can be provided according to users' request.

2. Technical Specifications:

Overview:

Item	Measurement Range	Humidity Accuracy	Temperature Accuracy	Resolution	Package
DHT11	20-90%RH 0-50 ℃	±5％RH	±2℃	1	4 Pin Single Row

Detailed Specifications:

Parameters	**Conditions**	**Minimum**	**Typical**	**Maximum**
Humidity				
Resolution		1%RH	1%RH	1%RH
			8 Bit	
Repeatability			±1%RH	
Accuracy	25℃		±4%RH	
	0-50℃			±5%RH
Interchangeability	Fully Interchangeable			
Measurement Range	0℃	30%RH		90%RH
	25℃	20%RH		90%RH
	50℃	20%RH		80%RH
Response Time (Seconds)	1/e(63%)25℃, 1m/s Air	6 S	10 S	15 S
Hysteresis			±1%RH	
Long-Term Stability	Typical		±1%RH/year	
Temperature				
Resolution		1℃	1℃	1℃
		8 Bit	8 Bit	8 Bit
Repeatability			±1℃	
Accuracy		±1℃		±2℃
Measurement Range		0℃		50℃
Response Time (Seconds)	1/e(63%)	6 S		30 S

3. Typical Application (Figure 1)

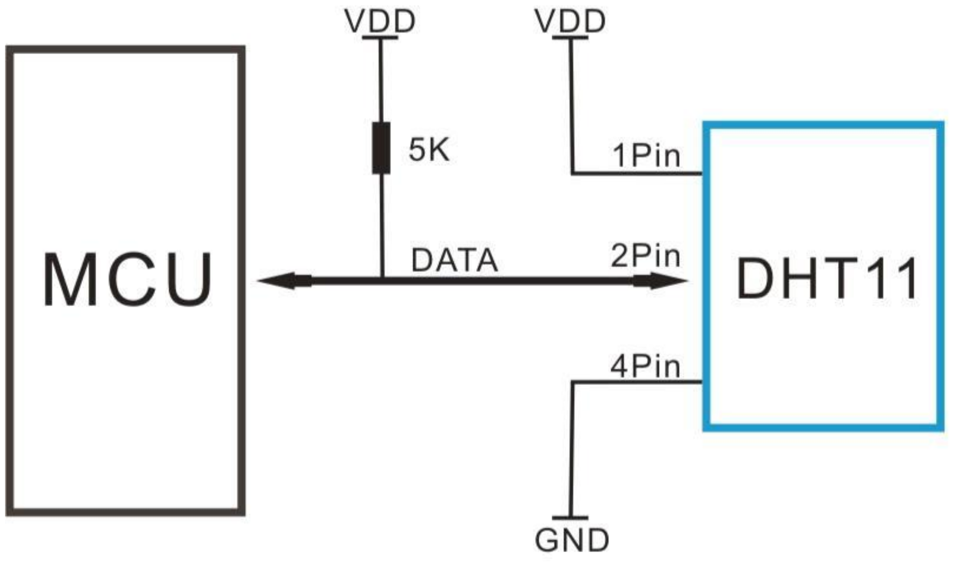


Figure 1 Typical Application

Note: 3Pin – Null; MCU = Micro-computer Unite or single chip Computer

When the connecting cable is shorter than 20 metres, a 5K pull-up resistor is recommended; when the connecting cable is longer than 20 metres, choose a appropriate pull-up resistor as needed.

4. Power and Pin

DHT11's power supply is 3-5.5V DC. When power is supplied to the sensor, do not send any instruction to the sensor in within one second in order to pass the unstable status. One capacitor valued 100nF can be added between VDD and GND for power filtering.

5. Communication Process: Serial Interface (Single-Wire Two-Way)

Single-bus data format is used for communication and synchronization between MCU and DHT11 sensor. One communication process is about 4ms.

Data consists of decimal and integral parts. A complete data transmission is **40bit**, and the sensor sends **higher data bit** first.

Data format: 8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data + 8bit check sum. If the data transmission is right, the check-sum should be the last 8bit of "8bit integral RH data + 8bit decimal RH data + 8bit integral T data + 8bit decimal T data".

5.1 Overall Communication Process (Figure 2, below)

When MCU sends a start signal, DHT11 changes from the low-power-consumption mode to the running-mode, waiting for MCU completing the start signal. Once it is completed, DHT11 sends a response signal of 40-bit data that include the relative humidity and temperature information to MCU. Users can choose to collect (read) some data. Without the start signal from MCU, DHT11 will not give the response signal to MCU. Once data is collected, DHT11 will change to the low-power-consumption mode until it receives a start signal from MCU again.

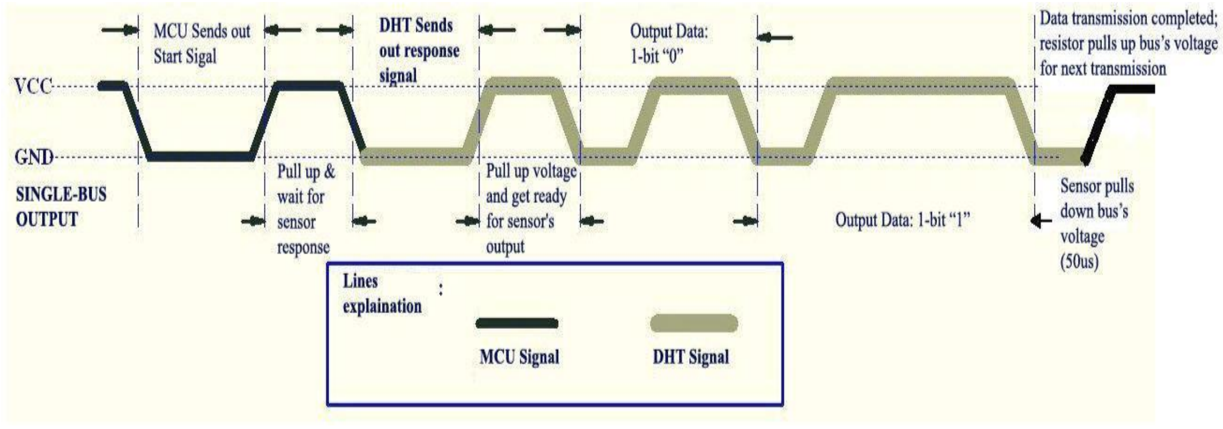

MCU Sends out Start Sigal
DHT Sends out response signal
Output Data: 1-bit "0"
Data transmission completed; resistor pulls up bus's voltage for next transmission
VCC
GND
SINGLE-BUS OUTPUT
Pull up & wait for sensor response
Pull up voltage and get ready for sensor's output
Output Data: 1-bit "1"
Sensor pulls down bus's voltage (50us)
Lines explaination :
MCU Signal
DHT Signal

Figure 2 Overall Communication Process

5.2 MCU Sends out Start Signal to DHT (Figure 3, below)

Data Single-bus free status is at high voltage level. When the communication between MCU and DHT11 begins, the programme of MCU will set Data Single-bus voltage level from high to low and this process must take at least 18ms to ensure DHT's detection of MCU's signal, then MCU will pull up voltage and wait 20-40us for DHT's response.

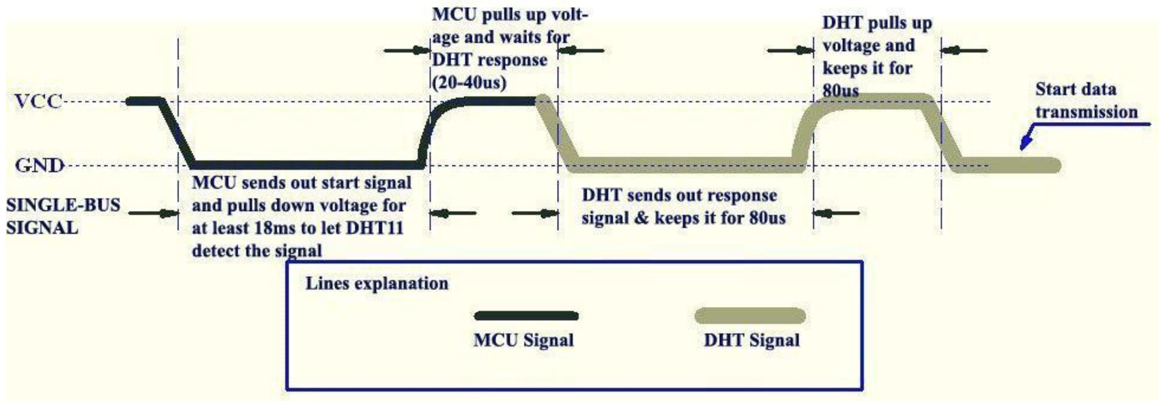

MCU pulls up voltage and waits for DHT response (20-40us)
DHT pulls up voltage and keeps it for 80us
Start data transmission
VCC
GND
SINGLE-BUS SIGNAL
MCU sends out start signal and pulls down voltage for at least 18ms to let DHT11 detect the signal
DHT sends out response signal & keeps it for 80us
Lines explanation
MCU Signal
DHT Signal

Figure 3 MCU Sends out Start Signal & DHT Responses

5.3 DHT Responses to MCU (Figure 3, above)

Once DHT detects the start signal, it will send out a low-voltage-level response signal, which lasts 80us. Then the programme of DHT sets Data Single-bus voltage level from low to high and keeps it for 80us for DHT's preparation for sending data.

When DATA Single-Bus is at the low voltage level, this means that DHT is sending the response signal. Once DHT sent out the response signal, it pulls up voltage and keeps it for 80us and prepares for data transmission.

When DHT is sending data to MCU, every bit of data begins with the 50us low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1" (see Figures 4 and 5 below).

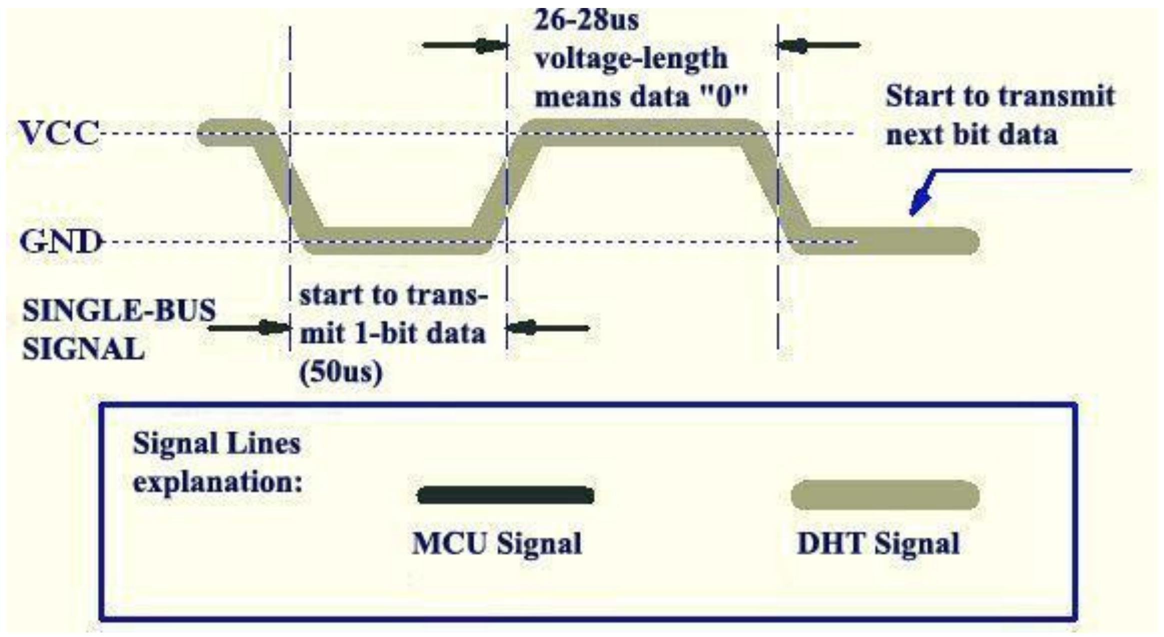


Figure 4 Data "0" Indication

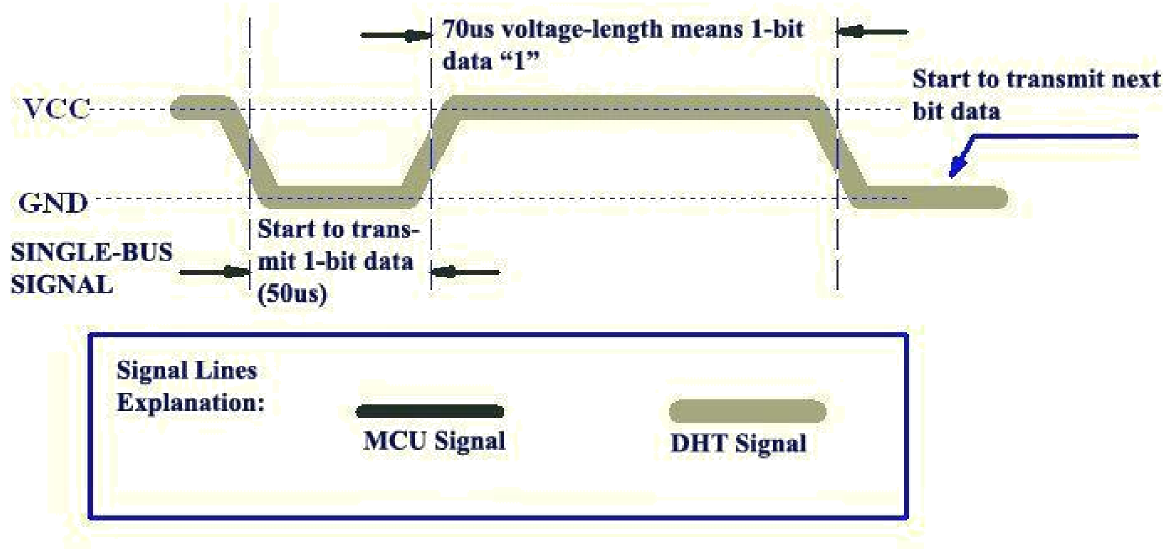

70us voltage-length means 1-bit data "1"
Start to transmit next bit data
VCC
GND
SINGLE-BUS SIGNAL
Start to trans-mit 1-bit data (50us)
Signal Lines Explanation:
MCU Signal
DHT Signal

Figure 5 Data "1" Indication

If the response signal from DHT is always at high-voltage-level, it suggests that DHT is not responding properly and please check the connection. When the last bit data is transmitted, DHT11 pulls down the voltage level and keeps it for 50us. Then the Single-Bus voltage will be pulled up by the resistor to set it back to the free status.

6. Electrical Characteristics

VDD=5V, T = 25℃ (unless otherwise stated)

	Conditions	Minimum	Typical	Maximum
Power Supply	DC	3V	5V	5.5V
Current Supply	Measuring	0.5mA		2.5mA
	Average	0.2mA		1mA
	Standby	100uA		150uA
Sampling period	Second	1		

Note: Sampling period at intervals should be no less than 1 second.

7. Attentions of application

(1) Operating conditions

Applying the DHT11 sensor beyond its working range stated in this datasheet can result in 3%RH signal shift/discrepancy. The DHT11 sensor can recover to the calibrated status gradually when it gets back to the normal operating condition and works within its range. Please refer to (3) of this section to accelerate its recovery. Please be aware that operating the DHT11 sensor in the non-normal working conditions will accelerate sensor's aging process.

(2) Attention to chemical materials

Vapor from chemical materials may interfere with DHT's sensitive-elements and debase its sensitivity. A high degree of chemical contamination can permanently damage the sensor.

(3) Restoration process when (1) & (2) happen

Step one: Keep the DHT sensor at the condition of Temperature 50~60Celsius, humidity <10%RH for 2 hours;

Step two:K keep the DHT sensor at the condition of Temperature 20~30Celsius, humidity >70%RH for 5 hours.

(4) Temperature Affect

Relative humidity largely depends on temperature. Although temperature compensation technology is used to ensure accurate measurement of RH, it is still strongly advised to keep the humidity and temperature sensors working under the same temperature. DHT11 should be mounted at the place as far as possible from parts that may generate heat.

(5) Light Affect

Long time exposure to strong sunlight and ultraviolet may debase DHT's performance.

(6) Connection wires

The quality of connection wires will affect the quality and distance of communication and high quality shielding-wire is recommended.

(7) Other attentions

* Welding temperature should be bellow 260Celsius and contact should take less than 10 seconds.
* Avoid using the sensor under dew condition.
* Do not use this product in safety or emergency stop devices or any other occasion that failure of DHT11 may cause personal injury.
* Storage: Keep the sensor at temperature 10-40 ℃, humidity <60%RH.

Declaim:

D-Robotics: www.droboticsonline.com

Email contact: d_robotics@hotmail.co.uk

Appendix B

BMP180

DIGITAL PRESSURE SENSOR

Key features

Pressure range:	300 ... 1100hPa (+9000m ... -500m relating to sea level)	
Supply voltage:	1.8 ... 3.6V (V_{DD})	
	1.62V ... 3.6V (V_{DDIO})	
Package:	LGA package with metal lid	
	Small footprint:	3.6mm x 3.8mm
	Super-flat:	0.93mm height
Low power:	5µA at 1 sample / sec. in standard mode	
Low noise:	0.06hPa (0.5m) in ultra low power mode	
	0.02hPa (0.17m) advanced resolution mode	

* Temperature measurement included
* I^2C interface
* Fully calibrated
* Pb-free, halogen-free and RoHS compliant,
* MSL 1

Typical applications

- Enhancement of GPS navigation (dead-reckoning, slope detection, etc.)
- In- and out-door navigation
- Leisure and sports
- Weather forecast
- Vertical velocity indication (rise/sink speed)

BMP180 general description

The BMP180 is the function compatible successor of the BMP085, a new generation of high precision digital pressure sensors for consumer applications.

The ultra-low power, low voltage electronics of the BMP180 is optimized for use in mobile phones, PDAs, GPS navigation devices and outdoor equipment. With a low altitude noise of merely 0.25m at fast conversion time, the BMP180 offers superior performance. The I^2C interface allows for easy system integration with a microcontroller.

The BMP180 is based on piezo-resistive technology for EMC robustness, high accuracy and linearity as well as long term stability.

Robert Bosch is the world market leader for pressure sensors in automotive applications. Based on the experience of over 400 million pressure sensors in the field, the BMP180 continues a new generation of micro-machined pressure sensors.

1. Electrical characteristics

If not stated otherwise, the given values are ±3-Sigma values over temperature/voltage range in the given operation mode. All values represent the new parts specification; additional solder drift is shown separately.

Table 1: Operating conditions, output signal and mechanical characteristics

Parameter	Symbol	Condition	Min	Typ	Max	Units
Operating temperature	T_A	operational	-40		+85	°C
		full accuracy	0		+65	
Supply voltage	V_{DD}	ripple max. 50mVpp	1.8	2.5	3.6	V
			1.62	2.5	3.6	
Supply current @ 1 sample / sec. 25°C	I_{DDLOW}	ultra low power mode		3		µA
	I_{DDSTD}	standard mode		5		µA
	I_{DDHR}	high resolution mode		7		µA
	I_{DDUHR}	Ultra high res. mode		12		µA
	I_{DDAR}	Advanced res. mode		32		µA
Peak current	I_{peak}	during conversion		650	1000	µA
Standby current	I_{DDSBM}	@ 25°C		0.1	4[1]	µA
Relative accuracy pressure $V_{DD} = 3.3V$		950 . . . 1050 hPa @ 25 °C		±0.12		hPa
				±1.0		m
		700 … 900hPa 25 . . . 40 °C		±0.12		hPa
				±1.0		m
Absolute accuracy pressure $V_{DD} = 3.3V$		300 . . . 1100 hPa 0 . . . +65 °C	-4.0	-1.0*	+2.0	hPa
		300 . . . 1100 hPa -20 . . . 0 °C	-6.0	-1.0*	+4.5	hPa
Resolution of output data		pressure		0.01		hPa
		temperature		0.1		°C
Noise in pressure		see table on page 12-13				
Absolute accuracy temperature $V_{DD} = 3.3V$		@ 25 °C	-1.5	±0.5	+1.5	°C
		0 . . . +65 °C	-2.0	±1.0	+2.0	°C

[1] at 85°C

Conversion time pressure	$t_{c\_p\_low}$	ultra low power mode		3	4.5	ms
	$t_{c\_p\_std}$	standard mode		5	7.5	ms
	$t_{c\_p\_hr}$	high resolution mode		9	13.5	ms
	$t_{c\_p\_luhr}$	ultra high res. mode		17	25.5	ms
	$t_{c\_p\_ar}$	Advanced res. mode		51	76.5	ms
Conversion time temperature	t_{C_temp}	standard mode		3	4.5	ms
Serial data clock	f_{SCL}				3.4	MHz
Solder drifts		Minimum solder height 50µm	-0.5		+2	hPa
Long term stability**		12 months		±1.0		hPa

* The typical value is: -1±1
** Long term stability is specified in the full accuracy operating pressure range 0 … 65°C

2. Absolute maximum ratings

Table 2: Absolute maximum ratings

Parameter	**Condition**	**Min**	**Max**	**Units**
Storage temperature		-40	+85	°C
Supply voltage	all pins	-0.3	+4.25	V
ESD rating	HBM, R = 1.5kΩ, C = 100pF		±2	kV
Overpressure			10,000	hPa

The BMP180 has to be handled as Electrostatic Sensitive Device (ESD).

Figure 1: ESD

3. Operation

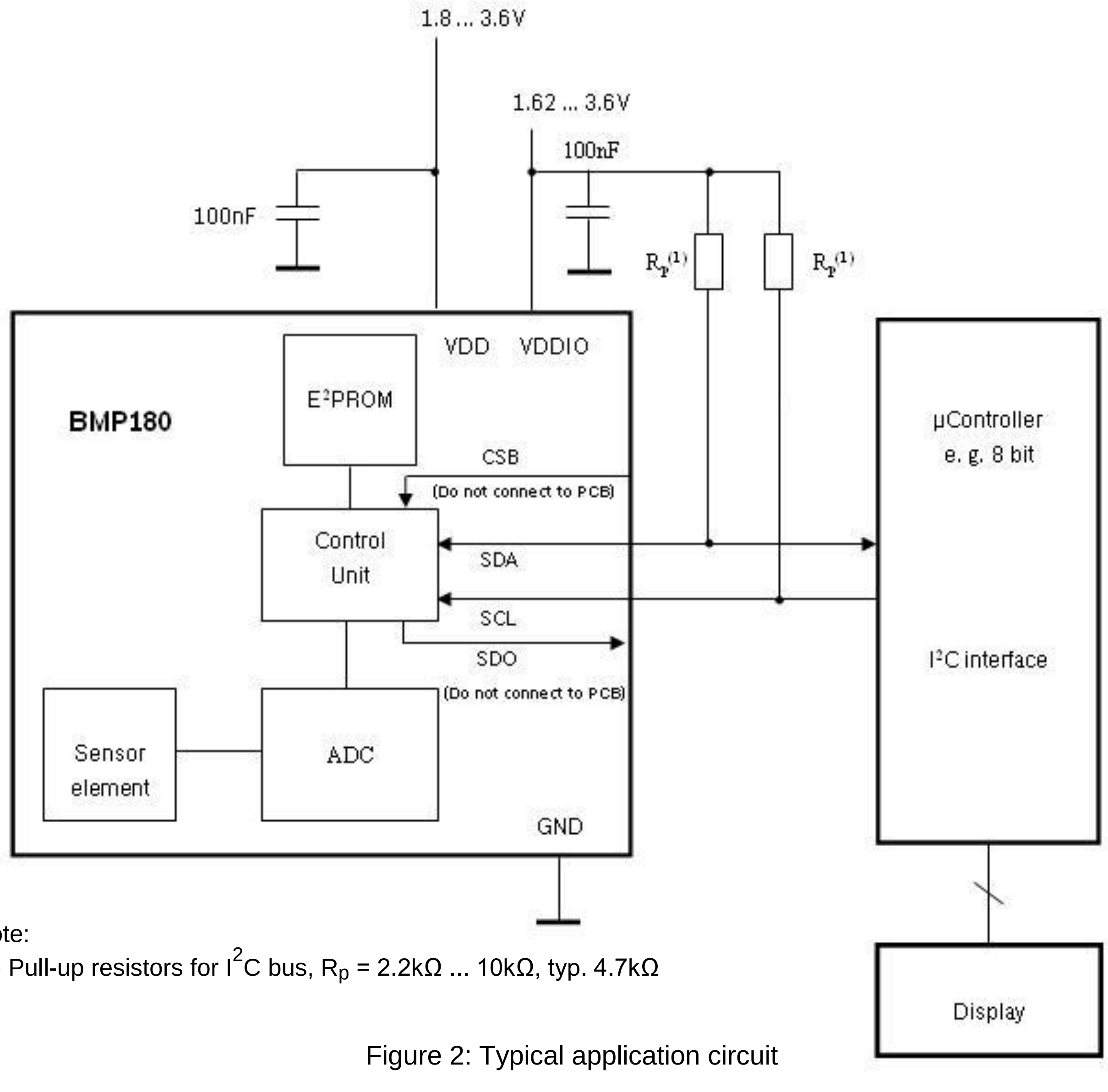
3.1 General description

The BMP180 is designed to be connected directly to a microcontroller of a mobile device via the I^2C bus. The pressure and temperature data has to be compensated by the calibration data of the E^2PROM of the BMP180.

3.2 General function and application schematics

The BMP180 consists of a piezo-resistive sensor, an analog to digital converter and a control unit with E^2PROM and a serial I^2C interface. The BMP180 delivers the uncompensated value of pressure and temperature. The E^2PROM has stored 176 bit of individual calibration data. This is used to compensate offset, temperature dependence and other parameters of the sensor.

- UP = pressure data (16 to 19 bit)
- UT = temperature data (16 bit)



Note:
(1) Pull-up resistors for I^2C bus, R_p = 2.2kΩ ... 10kΩ, typ. 4.7kΩ

Figure 2: Typical application circuit

3.3 Measurement of pressure and temperature

For all calculations presented here an ANSI C code is available from Bosch Sensortec ("BMP180 _API").

The microcontroller sends a start sequence to start a pressure or temperature measurement. After converting time, the result value (UP or UT, respectively) can be read via the I^2C interface. For calculating temperature in °C and pressure in hPa, the calibration data has to be used. These constants can be read out from the BMP180 E^2PROM via the I^2C interface at software initialization.

The sampling rate can be increased up to 128 samples per second (standard mode) for dynamic measurement. In this case, it is sufficient to measure the temperature only once per second and to use this value for all pressure measurements during the same period.

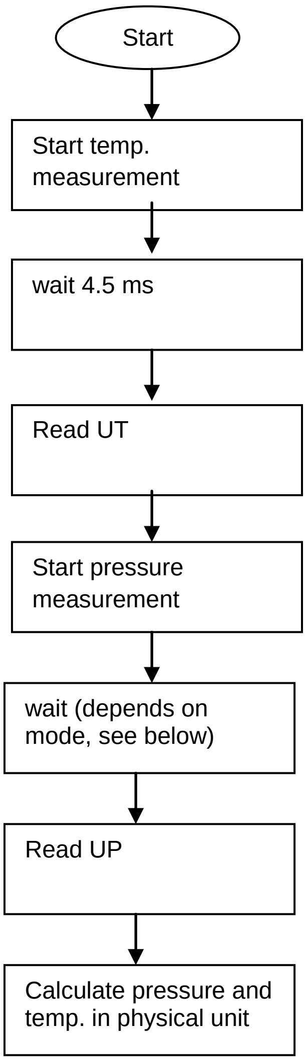

Start
Start temp. measurement
wait 4.5 ms
Read UT
Start pressure measurement
wait (depends on mode, see below)
Read UP
Calculate pressure and temp. in physical unit

Figure 3: Measurement flow BMP180

3.3.1 Hardware pressure sampling accuracy modes

By using different modes the optimum compromise between power consumption, speed and resolution can be selected, see below table.

Table 3: Overview of BMP180 hardware accuracy modes, selected by driver software via the variable *oversampling_setting*

Mode	**Parameter** ***oversampling_setting***	**Internal number of samples**	**Conversion time pressure max. [ms]**	**Avg. current @ 1 sample/s typ. [µA]**	**RMS noise typ. [hPa]**	**RMS noise typ. [m]**
ultra low power	0	1	4.5	3	0.06	0.5
standard	1	2	7.5	5	0.05	0.4
high resolution	2	4	13.5	7	0.04	0.3
ultra high resolution	3	8	25.5	12	0.03	0.25

For further information on noise characteristics see the relevant application note "Noise in pressure sensor applications".

All modes can be performed at higher speeds, e.g. up to 128 times per second for standard mode, with the current consumption increasing proportionally to the sample rate.

3.3.2 Software pressure sampling accuracy modes

For applications where a low noise level is critical, averaging is recommended if the lower bandwidth is acceptable. Oversampling can be enabled using the software API driver (with OSR = 3).

Table 4: Overview of BMP180 software accuracy mode, selected by driver software via the variable *software_oversampling_setting*

Mode	**Parameter** ***oversampling_setting***	**software_ oversampl ing_settin g**	**Conversion time pressure max. [ms]**	**Avg. current @ 1 sample/s typ. [µA]**	**RMS noise typ. [hPa]**	**RMS noise typ. [m]**
Advanced resolution	3	1	76.5	32	0.02	0.17

3.4 Calibration coefficients

The 176 bit E^2PROM is partitioned in 11 words of 16 bit each. These contain 11 calibration coefficients. Every sensor module has individual coefficients. Before the first calculation of temperature and pressure, the master reads out the E^2PROM data.
The data communication can be checked by checking that none of the words has the value 0 or 0xFFFF.

Table 5: Calibration coefficients

	BMP180 reg adr	
Parameter	**MSB**	**LSB**
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

3.5 Calculating pressure and temperature

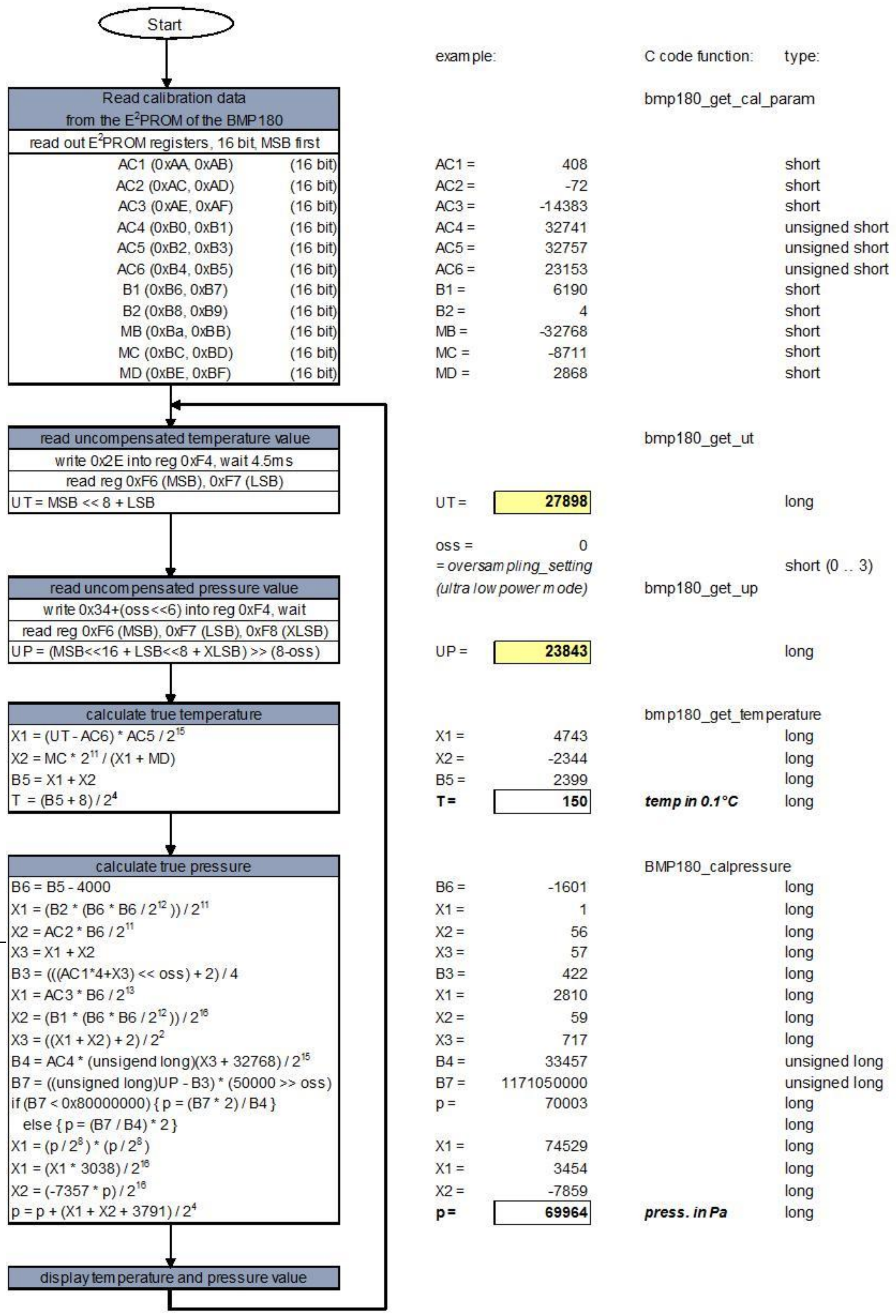
The mode (ultra low power, standard, high, ultra high resolution) can be selected by the variable *oversampling_setting* (0, 1, 2, 3) in the C code.

Calculation of true temperature and pressure in steps of 1Pa (= 0.01hPa = 0.01mbar) and temperature in steps of 0.1°C.

The following figure shows the detailed algorithm for pressure and temperature measurement.

This algorithm is available to customers as reference C source code ("BMP180_ API") from Bosch Sensortec and via its sales and distribution partners. **Please contact your Bosch Sensortec representative for details.**

Calculation of pressure and temperature for BMP180



Flowchart	example:		C code function:	type:
Read calibration data from the E^{2}PROM of the BMP180			bmp180_get_cal_param	
read out E^{2}PROM registers, 16 bit, MSB first				
AC1 (0xAA, 0xAB) (16 bit)	AC1 =	408		short
AC2 (0xAC, 0xAD) (16 bit)	AC2 =	-72		short
AC3 (0xAE, 0xAF) (16 bit)	AC3 =	-14383		short
AC4 (0xB0, 0xB1) (16 bit)	AC4 =	32741		unsigned short
AC5 (0xB2, 0xB3) (16 bit)	AC5 =	32757		unsigned short
AC6 (0xB4, 0xB5) (16 bit)	AC6 =	23153		unsigned short
B1 (0xB6, 0xB7) (16 bit)	B1 =	6190		short
B2 (0xB8, 0xB9) (16 bit)	B2 =	4		short
MB (0xBa, 0xBB) (16 bit)	MB =	-32768		short
MC (0xBC, 0xBD) (16 bit)	MC =	-8711		short
MD (0xBE, 0xBF) (16 bit)	MD =	2868		short
read uncompensated temperature value			bmp180_get_ut	
write 0x2E into reg 0xF4, wait 4.5ms				
read reg 0xF6 (MSB), 0xF7 (LSB)				
UT = MSB << 8 + LSB	UT =	**27898**		long
	oss =	0		
	= oversampling_setting			short (0 .. 3)
read uncompensated pressure value	*(ultra low power mode)*		bmp180_get_up	
write 0x34+(oss<<6) into reg 0xF4, wait				
read reg 0xF6 (MSB), 0xF7 (LSB), 0xF8 (XLSB)				
UP = (MSB<<16 + LSB<<8 + XLSB) >> (8-oss)	UP =	**23843**		long
calculate true temperature			bmp180_get_temperature	
X1 = (UT - AC6) * AC5 / 2^{15}	X1 =	4743		long
X2 = MC * 2^{11} / (X1 + MD)	X2 =	-2344		long
B5 = X1 + X2	B5 =	2399		long
T = (B5 + 8) / 2^4	**T =**	**150**	***temp in 0.1°C***	long
calculate true pressure			BMP180_calpressure	
B6 = B5 - 4000	B6 =	-1601		long
X1 = (B2 * (B6 * B6 / 2^{12})) / 2^{11}	X1 =	1		long
X2 = AC2 * B6 / 2^{11}	X2 =	56		long
X3 = X1 + X2	X3 =	57		long
B3 = (((AC1*4+X3) << oss) + 2) / 4	B3 =	422		long
X1 = AC3 * B6 / 2^{13}	X1 =	2810		long
X2 = (B1 * (B6 * B6 / 2^{12})) / 2^{16}	X2 =	59		long
X3 = ((X1 + X2) + 2) / 2^2	X3 =	717		long
B4 = AC4 * (unsigend long)(X3 + 32768) / 2^{15}	B4 =	33457		unsigned long
B7 = ((unsigned long)UP - B3) * (50000 >> oss)	B7 =	1171050000		unsigned long
if (B7 < 0x80000000) { p = (B7 * 2) / B4 }	p =	70003		long
else { p = (B7 / B4) * 2 }				long
X1 = (p / 2^8) * (p / 2^8)	X1 =	74529		long
X1 = (X1 * 3038) / 2^{16}	X1 =	3454		long
X2 = (-7357 * p) / 2^{16}	X2 =	-7859		long
p = p + (X1 + X2 + 3791) / 2^4	**p =**	**69964**	***press. in Pa***	long
display temperature and pressure value				

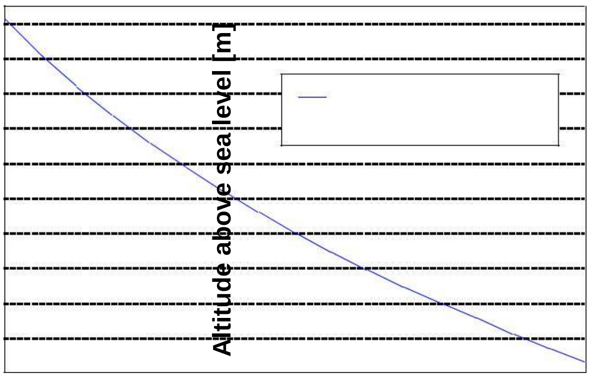
Figure 4: Algorithm for pressure and temperature measurement

3.6 Calculating absolute altitude

With the measured pressure p and the pressure at sea level p_0 e.g. 1013.25hPa, the altitude in meters can be calculated with the international barometric formula:

$$\text{altitude} = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

Thus, a pressure change of Δp = 1hPa corresponds to 8.43m at sea level.



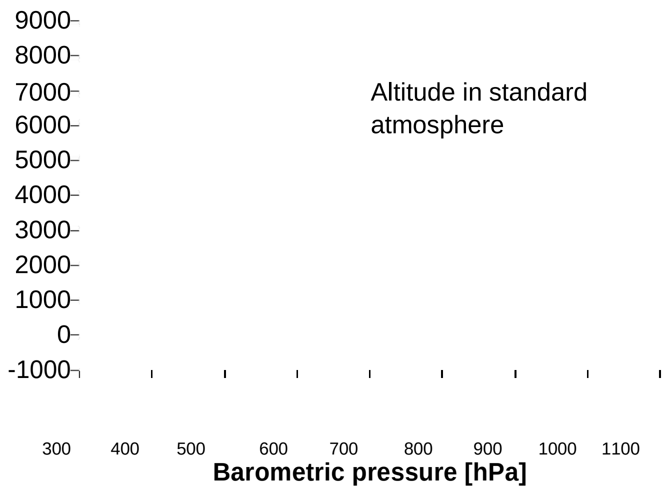


Figure 5**:** Transfer function: Altitude over sea level – Barometric pressure

3.7 Calculating pressure at sea level

With the measured pressure *p* and the absolute altitude the pressure at sea level can be calculated:

$$p_0 = \frac{p}{\left(1 - \frac{\text{altitude}}{44330}\right)^{5.255}}$$

Thus, a difference in altitude of Δaltitude = 10m corresponds to 1.2hPa pressure change at sea level.

4. Global Memory Map

The memory map below shows all externally accessible data registers which are needed to operate BMP180. The left columns show the memory addresses. The columns in the middle depict the content of each register bit. The colors of the bits indicate whether they are read-only, write-only or read- and writable. The memory is volatile so that the writable content has to be re-written after each power-on.
Not all register addresses are shown. These registers are reserved for further Bosch factory testing and trimming.

Register Name	**Register Adress**	**bit7**	**bit6**	**bit5**	**bit4**	**bit3**	**bit2**	**bit1**	**bit0**	**Reset state**
out_xlsb	F8h	adc_out_xlsb<7:3>					0	0	0	00h
out_lsb	F7h	adc_out_lsb<7:0>								00h
out_msb	F6h	adc_out_msb<7:0>								80h
ctrl_meas	F4h	oss<1:0>		sco	measurement control					00h
soft reset	E0h	reset								00h
id	D0h	id<7:0>								55h
calib21 downto calib0	BFh *down* to AAh	calib21<7:0> *down* to calib0<7:0>								n/a

Registers:	Control registers	Calibration registers 🟩	Data registers 🟨	Fixed ▨
Type:	read / write	read only	read only	read only

Figure 6: Memory map

Measurement control (register F4h <4:0>): Controls measurements. Refer to Figure 6 for usage details.

Sco (register F4h <5>): Start of conversion. The value of this bit stays “1” during conversion and is reset to “0” after conversion is complete (data registers are filled).

Oss (register F4h <7:6>): controls the oversampling ratio of the pressure measurement (00b: single, 01b: 2 times, 10b: 4 times, 11b: 8 times).

Soft reset (register E0h): Write only register. If set to 0xB6, will perform the same sequence as power on reset.

Chip-id (register D0h): This value is fixed to 0x55 and can be used to check whether communication is functioning.

After conversion, data registers can be read out in any sequence (i.e. MSB first or LSB first). Using a burst read is not mandatory.

5. I^{2}C Interface

- I^{2}C is a digital two wire interface
- Clock frequencies up to 3.4Mbit/sec. (I^{2}C standard, fast and high-speed mode supported)
- SCL and SDA needs a pull-up resistor, typ. 4.7kOhm to V_{DDIO} (one resistor each for all the I^{2}C bus)

The I^{2}C bus is used to control the sensor, to read calibration data from the E^{2}PROM and to read the measurement data when A/D conversion is finished. SDA (serial data) and SCL (serial clock) have open-drain outputs.

For detailed I^{2}C-bus specification please refer to:
http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

5.1 I^{2}C specification

Table 6: Electrical parameters for the I^{2}C interface

Parameter	Symbol	Min.	Typ	Max.	Units
Clock input frequency	f_{SCL}			3.4	MHz
Input-low level	V_{IL}	0		0.2 * V_{DDIO}	V

Input-high level	V_{IH}	0.8 * V_{DDIO}		V_{DDIO}	V
Voltage output low level @ V_{DDIO} = 1.62V, I_{OL} = 3mA	V_{OL}			0.3	V
SDA and SCL pull-up resistor	$R_{pull-up}$	2.2		10	kOhm
SDA sink current @ V_{DDIO} = 1.62V, V_{OL} = 0.3V	I_{SDA_sink}		9		mA
Start-up time after power-up, before first communication	t_{Start}	10			Ms

5.2 Device and register address

The BMP180 module address is shown below. The LSB of the device address distinguishes between read (1) and write (0) operation, corresponding to address 0xEF (read) and 0xEE (write).

Table 7: BMP180 addresses

A7	A6	A5	A4	A3	A2	A1	W/R
1	1	1	0	1	1	1	0/1

5.3 I^2C protocol

The I^2C interface protocol has special bus signal conditions. Start (S), stop (P) and binary data conditions are shown below. At start condition, SCL is high and SDA has a falling edge. Then the slave address is sent. After the 7 address bits, the direction control bit R/W selects the read or write operation. When a slave device recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.

At stop condition, SCL is also high, but SDA has a rising edge. Data must be held stable at SDA when SCL is high. Data can change value at SDA only when SCL is low.

Even though V_{DDIO} can be powered on before V_{DD}, there is a chance of excessive power consumption (a few mA) if this sequence is used, and the state of the output pins is undefined so

that the bus can be locked. Therefore, V_{DD} *must* be powered before

V_{DDIO} unless the limitations above are understood and not critical.

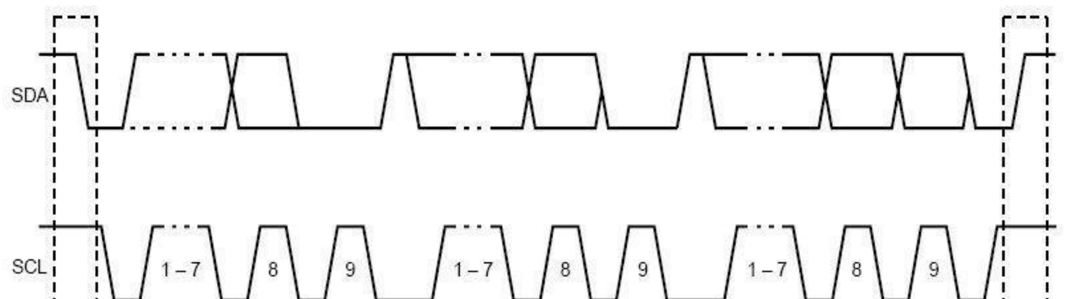


Figure 7: I^2C protocol

5.4 Start temperature and pressure measurement

The timing diagrams to start the measurement of the temperature value UT and pressure value UP are shown below. After start condition the master sends the device address write, the register address and the control register data. The BMP180 sends an acknowledgement (ACKS) every 8 data bits when data is received. The master sends a stop condition after the last ACKS.

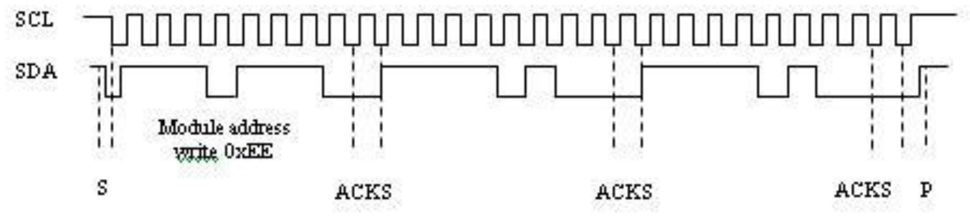


Figure 8: Timing diagram for starting pressure measurement

Abbreviations:

S	Start
P	Stop
ACKS	Acknowledge by Slave
ACKM	Acknowledge by Master
NACKM	Not Acknowledge by Master

5.5 Read A/D conversion result or E^{2}PROM data

To read out the temperature data word UT (16 bit), the pressure data word UP (16 to 19 bit) and the E^{2}PROM data proceed as follows:

After the start condition the master sends the module address write command and register address. The register address selects the read register:

E^{2}PROM data registers	0xAA to 0xBF
Temperature or pressure value UT or UP	0xF6 (MSB), 0xF7 (LSB), optionally 0xF8 (XLSB)

Then the master sends a restart condition followed by the module address read that will be acknowledged by the BMP180 (ACKS). The BMP180 sends first the 8 MSB, acknowledged by the master (ACKM), then the 8 LSB. The master sends a "not acknowledge" (NACKM) and finally a stop condition.

Optionally for ultra high resolution, the XLSB register with address 0xF8 can be read to extend the 16 bit word to up to 19 bits; refer to the application programming interface (API) software rev. 1.1 (“BMP180_ API”, available from Bosch Sensortec).

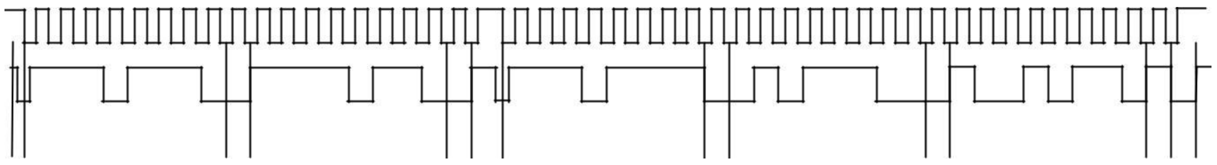


Figure 9: Timing diagram read 16 bit A/D conversion result

6. Package

6.1 Pin configuration

Picture shows the device in top view. Device pins are shown here transparently only for orientation purposes.

Figure 10: Layout pin configuration BMP180

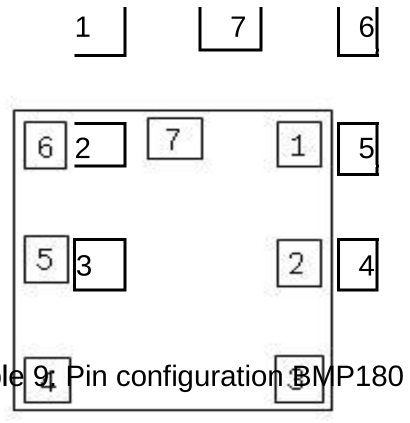


Table 9: Pin configuration BMP180

in No	Name	Function
1	CSB*	Chip select
2	VDD	Power supply
3	VDDIO	Digital power supply
4	SDO*	SPI output
5	SCL	I2C serial bus clock input
6	SDA	I2C serial bus data (or SPI input)
7	GND	Ground

* A pin compatible product variant with SPI interface is possible upon customer's request. For I^2C (standard case) CSB and SDO are not used, they have to be left open.
All pins have to be soldered to the PCB for symmetrical stress input even though they are not connected internally.

6.2 Outline dimensions

The sensor housing is a 7Pin LGA package with metal lid. Its dimensions are 3.60mm (±0.1 mm) x 3.80mm (±0.1 mm) x 0.93mm (±0.07 mm).

Note: All dimensions are in mm.

6.2.1 Bottom view

0,50

↑ 0,60

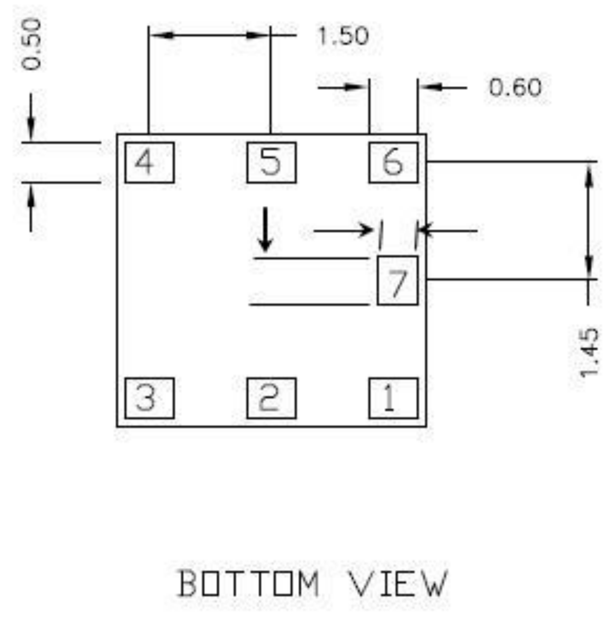


Figure 11: Bottom view BMP180

6.2.2 Top view

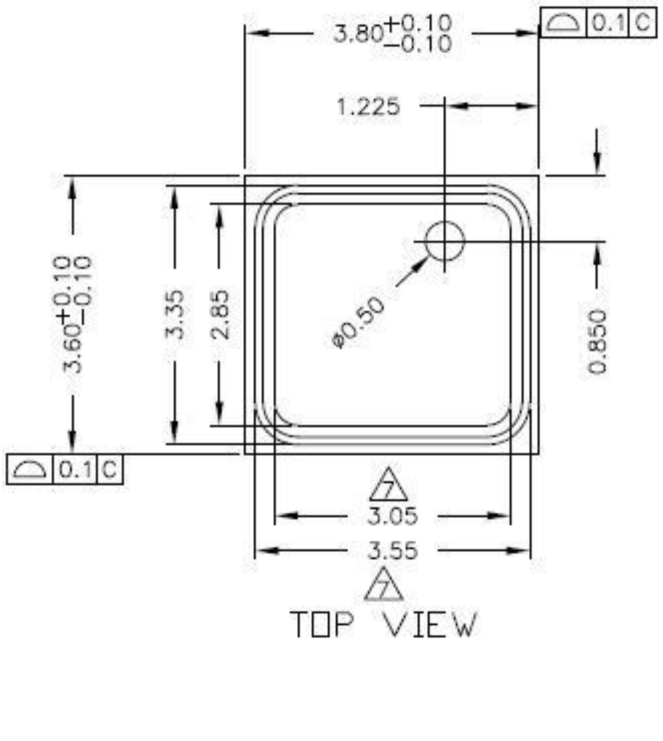


Figure 12: Top view BMP180

6.2.3 Side view

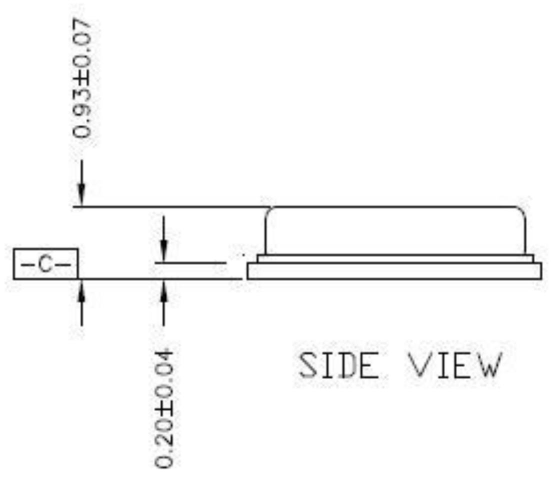


Figure 13: Side view BMP180

6.3 Moisture sensitivity level and soldering

The BMP180 is classified MSL 1 (moisture sensitivity level) according to IPC/JEDEC standards J-STD-020D and J-STD-033A.

The device can be soldered Pb-free with a peak temperature of 260°C for 20 to 40 sec. The minimum height of the solder after reflow shall be at least 50µm. This is required for good mechanical decoupling between the sensor device and the printed circuit board (PCB).

To ensure good solder-ability, the devices shall be stored at room temperature (20°C).

The soldering process can lead to an offset shift.

6.4 RoHS compliancy

The BMP180 sensor meets the requirements of the EC directive "Restriction of hazardous substances (RoHS)", please refer also to:

"Directive 2002/95/EC of the European Parliament and of the Council of 27 January 2003 on the restriction of the use of certain hazardous substances in electrical and electronic equipment".

The BMP180 sensor is also halogen-free.

6.5 Mounting and assembly recommendations

In order to achieve the specified performance for you design, the following recommendations and the
“Handling, soldering & mounting instructions BMP180” should be taken into consideration
when mounting a pressure sensor on a printed-circuit board (PCB):

- The clearance above the metal lid shall be 0.1mm at minimum.

- For the device housing appropriate venting needs to be provided in case the ambient pressure shall be measured.

- Liquids shall not come into direct contact with the device.

- During operation the sensor is sensitive to light, which can influence the accuracy of the measurement (photo-current of silicon).

- The BMP180 shall not the placed close the fast heating parts. In case of gradients > 3°C/sec. it is recommended to follow Bosch Sensortec application note ANP015, "Correction of errors induced by fast temperature changes". Please contact your Bosch Sensortec representative for details.

7. Legal disclaimer

7.1 Engineering samples

Engineering Samples are marked with an asterisk (*) or (e). Samples may vary from the valid technical specifications of the product series contained in this data sheet. They are therefore not intended or fit for resale to third parties or for use in end products. Their sole purpose is internal client testing. The testing of an engineering sample may in no way replace the testing of a product series. Bosch Sensortec assumes no liability for the use of engineering samples. The Purchaser shall indemnify Bosch Sensortec from all claims arising from the use of engineering samples.

7.2 Product use

Bosch Sensortec products are developed for the consumer goods industry. They may only be used within the parameters of this product data sheet. They are not fit for use in life-sustaining or security sensitive systems. Security sensitive systems are those for which a malfunction is expected to lead to bodily harm or significant property damage. In addition, they are not fit for use in products which interact with motor vehicle systems.

The resale and/or use of products are at the purchaser's own risk and his own responsibility. The examination of fitness for the intended use is the sole responsibility of the Purchaser.

The purchaser shall indemnify Bosch Sensortec from all third party claims arising from any product use not covered by the parameters of this product data sheet or not approved by Bosch Sensortec and reimburse Bosch Sensortec for all costs in connection with such claims.

The purchaser must monitor the market for the purchased products, particularly with regard to product safety, and inform Bosch Sensortec without delay of all security relevant incidents.

7.3 Application examples and hints

With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Bosch Sensortec hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights or copyrights of any third party. The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. They are provided for illustrative purposes only and no evaluation regarding infringement of intellectual property rights or copyrights or regarding functionality, performance or error has been made.

Appendix C

SIM900

SIMCom presents an ultra compact and reliable wireless module-SIM900. This is a complete Quad-band GSM/GPRS module in a SMT type and designed with a very powerful single-chip processor integrating AMR926EJ-S core, allowing you to benefit from small dimensions and cost-effective solutions.

Featuring an industry-standard interface, the SIM900 delivers GSM/GPRS 850/900/1800/1900MHz performance for voice, SMS, Data, and Fax in a small form factor and with low power consumption. With a tiny configuration of 24mm x 24mm x 3 mm, SIM900 can fit almost all the space requirements in your M2M applications, especially for slim and compact demands of design.

Smart Machine Smart Decision

General features

•Quad-Band 850/ 900/ 1800/ 1900 MHz
•GPRS multi-slot class 10/8
•GPRS mobile station class B
•Compliant to GSM phase 2/2+
 – Class 4 (2 W @850/ 900 MHz)
 – Class 1 (1 W @ 1800/1900MHz)
•Dimensions: 24* 24 * 3 mm
•Weight: 3.4g
•Control via AT commands (GSM 07.07 ,07.05 and SIMCOM enhanced AT Commands)
•SIM application toolkit
•Supply voltage range : 3.1 ... 4.8V
•Low power consumption: 1.5mA(sleep mode)
•Operation temperature: -40° C to +85 ° C

Specifications for Fax

•Group 3, class 1

Specifications for Data

•GPRS class 10: max. 85.6 kbps (downlink)
•PBCCH support
•Coding schemes CS 1, 2, 3, 4
•CSD up to 14.4 kbps
•USSD
•Non transparent mode
•PPP-stack

Specifications for SMS via GSM/GPRS

•Point to point MO and MT
•SMS cell broadcast
•Text and PDU mode

Software features

•0710 MUX protocol
•embedded TCP/UDP protocol
•FTP/HTTP(available at July ,2010)
•FOTA (available at July ,2010)
•MMS (available at July ,2010)
•Embedded AT (available at Q3,2010)

Specifications for Voice

•Tricodec
– Half rate (HR)
– Full rate (FR)
– Enhanced Full rate (EFR)
•Hands-free operation （Echo suppression）
•AMR
© Half rate (HR)
© Full rate (FR)

Interfaces

•Interface to external SIM 3V/ 1.8V
•analog audio interface
•RTC backup
•SPI interface (option)
•Serial interface
•Antenna pad
•I2C
•GPIO
•PWM
•ADC

Compatibility

•AT cellular command interface

Certificates:

•CE
•FCC
•ROHS
•PTCRB
•GCF
•IC
•ICASA
•TA
•REACH

Certificates (on going):

•AT&T(will be finished at end of July 2010)

More about SIMCom SIM900 Please contact:
Tel: 86-21-32523300
Fax: 86-21-32523301 Email: simcom@sim.com Website: www.sim.com/wm

Appendix D

THE Code of The system

```
#include <SoftwareSerial.h>                          //declare the library of the serial port
#define terminator 10                               // DEC value for a LF(line feed) to skip while loop
#include <dht.h>                                      //declare the library of the dht sensor
#include <SFE_BMP180.h>                              //declare the library of the BMP sensor
#include <Wire.h>                                    //declare the library of wires
dht DHT;                                            //declare the variable of the dht sensor
#define DHT11_PIN 3                                 // declare the pin of the dht sensor
SFE_BMP180 pressure;
#define ALTITUDE 990.0
double baseline;                                   // baseline pressure
float val;                                         // declare a variable from a float type
int tempPin = 0;                                   //make the initial value of the sensor zero
float mv;                                         // declare a variable from a float type
float dh;                                         // declare a variable from a float type
float pre;                                       // declare a variable from a float type
char ch[20]="wsp";                               // declare a variable from a character type
String IncDataSerial = "";                       // declare a variable from a string type
void setup()                                    //the main function
{
//  delay(1000);
  Serial.begin(19200);
  Serial1.begin(19200);
 if (pressure.begin())
    Serial.println("BMP180 init success");
  else
  {
    // Oops, something went wrong, this is usually a connection problem,
    Serial.println("BMP180 init fail\n\n");
    while(1);                                   // Pause forever.
  }
  baseline = getPressure();                    //read the value of the pressure sensor
    Serial.print("baseline pressure: ");       //print the value of the pressure sensor
  Serial.print(baseline);
  Serial.println(" mb");
                                             // Automatically power up the SIM900.
  pinMode(9, OUTPUT);                          //declare the mode of the pin number 9
  digitalWrite(9,LOW);                        //make the value of the ninth pin zero
  delay(1000);
  digitalWrite(9,HIGH);                       //make the value of the ninth pin 1
  delay(2500);
  digitalWrite(9,LOW);                        //make the value of the ninth pin zero
  delay(3500);
  // End of SIM900 power up.
  }
```

```
void loop()
{
  double a,P;                                                          //declare varibels
                  // Get a new pressure reading:
  P = getPressure();
  // a=getPressure(T);
//  a=getTemperature();
  /*char status;
  double T,P,p0,a;                                        //declare variables from double type
   status = pressure.startTemperature();      //read the reading of the pressure sensor
  if (status != 0)
  {
    delay(status);
status = pressure.getTemperature(T);        //read the reading of the temprature sensor
    if (status != 0)
    {
      // Print out the measurement:
      Serial.print("temperature: ");
      Serial.print(T,2);
       status = pressure.startPressure(3);
      if (status != 0)
      {
        delay(status);
         status = pressure.getPressure(P,T);
        if (status != 0)
        {
          // Print out the measurement:
          Serial.print("absolute pressure: ");
          Serial.print(P,2);
          Serial.print(" mb, ");
        }
         else Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");

  delay(5000);                                                                    // Pause for 5 seconds.
*/
    if (Serial1.available()>0)                                        // if date is coming from software serial port
==> data is coming from gprs shield
  {
    Boolean getLF = false;                                          //declare a variable from Boolean type as false

    while(Serial1.available()>0 && !getLF)               // reading data into string if activity is on port
and getLF is false ==> no LF have been send
    {
      char buffer=Serial1.read();                               // writing data into char
      IncDataSerial += buffer;                               //put the reading of the gprs in the buffer variable
and decrement it
```

```
if (buffer == terminator) {
        getLF = true;
      }
    }

    Serial.print(IncDataSerial);                          // send string ( char array ) to hardware serial
    Serial.print("\r");                                   // send a CR because it is missing
    IncDataSerial = "";

  }
reedtemp();                                              //declare the function of reading the value of the
temperature sensor

  //sendToServer(mv,dh, 20, ch, 30, 40) ; //send the values of sensors to the server
 sendToServer(mv,dh,baseline);
   if (Serial.available()>0)                        // if data is available on hardwareserial port ==> data is
comming from PC or notebook
    Serial1.write(Serial.read());                 // write it to the GPRS shield
}
void reedtemp()                                     //the function of reading the value of the temperature sensor
{
/* val = analogRead(tempPin);           //reading the temprature sensor
//Serial.println(val);                         //print the last reading
 mv = ((( val/1023.0)*5000)/10);       //convert the reading to real one
 Serial.println(mv);                           //print the real value
  delay(15000);
  Serial.println(mv);
  */
   int chk=DHT.read11(DHT11_PIN);
 //Serial.print("TEM=");
mv=(DHT.temperature);

  Serial.println(mv);
 Serial.println("hum=");

  dh=(DHT.humidity);
    Serial.print(dh);
    baseline = getPressure();
    Serial.print("baseline pressure: ");
  Serial.print(baseline);
Serial.println(" mb");
  delay(120000);
 /*
  char status;
  double T,P,p0,a;
   status = pressure.startTemperature();
  if (status != 0)
  {
    delay(status);
status = pressure.getTemperature(T);
    if (status != 0)
    {

      // Print out the measurement:
      Serial.print("temperature: ");
```

```
       status = pressure.startPressure(3);
      if (status != 0)
      {
        delay(status);
         status = pressure.getPressure(P,T);
        if (status != 0)
        {
          // Print out the measurement:
          Serial.print("absolute pressure: ");
        // Serial.print(P,2);
          pre= Serial.print(P,2);
          //Serial.print(pre);
          Serial.print(" mb, ");
          Serial.print(pre);
        }
         else Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");

  delay(5000);  // Pause for 5 seconds.
  */

  }
  //the function that send the data to the server
//there is an appendix and a paragraph in the report that interprets the AT commands
  void sendToServer(double _temp, double moins, double pres){

char temp[20];
Serial1.println("AT");
 delay(500);
Serial1.println("AT+CGATT?");
  delay(500);
Serial1.println("AT+SAPBR=3,1,\"APN\",\"wap\"");
  delay(500);
Serial1.println("AT+SAPBR=1,1");
  delay(500);
  Serial1.println("AT+HTTPINIT");
   delay(500);
 Serial1.println("AT+HTTPPARA=CID,1");
   delay(500);
    char str[200] = ("AT+HTTPPARA= \"URL\" ,http://weather.ppu.edu/receive_data_GET.php?");
   strcat(str,"location=");
   strcat(str,"location1");
   strcat(str,"&password=");
   strcat(str,"123123");
   strcat(str,"&temperature=");
   //strcat(str,temp);
dtostrf(_temp, 4, 4, temp);
   strcat(str,temp);
  // strcat(str,"&humidity=");
  //    strcat(str,temp);
strcat(str,"&humidity=");
dtostrf(moins, 4, 4, temp);
  // strcat(str,"&humidity=");
   strcat(str,temp);
//dtostrf(windVal, 4, 4, temp);
   strcat(str,"&pressure=");
      //strcat(str,temp);

   dtostrf(pres, 4, 4, temp);
   strcat(str,temp);
   //strcat(str,"&windDir=");
   //strcat(str,windDir);
//dtostrf(pres, 4, 4, temp);
  // strcat(str,"&pres=");
   //strcat(str,temp);
//dtostrf(rain, 4, 4, temp);
   //strcat(str,"&rain=");
   //strcat(str,temp);
  Serial1.println(str);
  delay(1000);
   // set http action type 0 = GET, 1 = POST, 2 = HEAD
  Serial1.println("AT+HTTPACTION=1");
   delay(1000);
 Serial1.println("AT+HTTPDATA=15000,15000");
 }
  /*void sendToServer(double _temp, double moins, double windVal, char windDir[20], double pres,
double rain){true

char temp[20];
Serial1.println("AT");
 delay(500);
Serial1.println("AT+CGATT?");
  delay(500);
Serial1.println("AT+SAPBR=3,1,\"APN\",\"wap\"");
  delay(500);
Serial1.println("AT+SAPBR=1,1");
  delay(500);
  Serial1.println("AT+HTTPINIT");
   delay(500);
 Serial1.println("AT+HTTPPARA=CID,1");
   delay(500);
    char str[200] = ("AT+HTTPPARA= \"URL\" ,http://weather.ppu.edu/request.php?");
   strcat(str,"location=");
   strcat(str,"position2");
   strcat(str,"&temp=");
dtostrf(_temp, 4, 4, temp);
   strcat(str,temp);
dtostrf(moins, 4, 4, temp);
   strcat(str,"&moins=");
   strcat(str,temp);
dtostrf(windVal, 4, 4, temp);
   strcat(str,"&windVal=");
```

```
       status = pressure.startPressure(3);
      if (status != 0)
      {
        delay(status);
         status = pressure.getPressure(P,T);
        if (status != 0)
        {
          // Print out the measurement:
          Serial.print("absolute pressure: ");
        // Serial.print(P,2);
          pre= Serial.print(P,2);
          //Serial.print(pre);
          Serial.print(" mb, ");
          Serial.print(pre);
        }
         else Serial.println("error retrieving pressure measurement\n");
      }
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");

  delay(5000);  // Pause for 5 seconds.
  */

  }

  void sendToServer(double _temp, double moins, double pres){

char temp[20];
Serial1.println("AT");
 delay(500);
Serial1.println("AT+CGATT?");
  delay(500);
Serial1.println("AT+SAPBR=3,1,\"APN\",\"wap\"");
  delay(500);
Serial1.println("AT+SAPBR=1,1");
  delay(500);
  Serial1.println("AT+HTTPINIT");
   delay(500);
 Serial1.println("AT+HTTPPARA=CID,1");
   delay(500);
    char str[200] = ("AT+HTTPPARA= \"URL\" ,http://weather.ppu.edu/receive_data_GET.php?");
   strcat(str,"location=");
   strcat(str,"location1");
   strcat(str,"&password=");
   strcat(str,"123123");
   strcat(str,"&temperature=");
   //strcat(str,temp);
dtostrf(_temp, 4, 4, temp);
   strcat(str,temp);
  // strcat(str,"&humidity=");
```

```
       strcat(str,temp);
   strcat(str,"&windDir=");
   strcat(str,windDir);
dtostrf(pres, 4, 4, temp);
   strcat(str,"&pres=");
   strcat(str,temp);
dtostrf(rain, 4, 4, temp);
   strcat(str,"&rain=");
   strcat(str,temp);
  Serial1.println(str);
  delay(1000);
     //delay(5000);
 // set http action type 0 = GET, 1 = POST, 2 = HEAD
  Serial1.println("AT+HTTPACTION=0");
   delay(1000);
 Serial1.println("AT+HTTPDATA=15000,15000");
 }*/
 /* void sendToServer(double _temp, double moins, double windVal, char windDir[20], double pres,
double rain){

   strcat(str,"location=");
   strcat(str,"position2");
   strcat(str,"&temp=");
dtostrf(_temp, 4, 4, temp);
   strcat(str,temp);
dtostrf(moins, 4, 4, temp);
   strcat(str,"&moins=");
   strcat(str,temp);
dtostrf(windVal, 4, 4, temp);
   strcat(str,"&windVal=");
   strcat(str,temp);
   strcat(str,"&windDir=");
   strcat(str,windDir);
dtostrf(pres, 4, 4, temp);
   strcat(str,"&pres=");
   strcat(str,temp);
dtostrf(rain, 4, 4, temp);
   strcat(str,"&rain=");
   strcat(str,temp);
  Serial1.println(str);
  delay(1000);

     //delay(5000);
 // set http action type 0 = GET, 1 = POST, 2 = HEAD
  Serial1.println("AT+HTTPACTION=1");
   delay(1000);
 Serial1.println("AT+HTTPDATA=15000,15000");
 }*/
 double getPressure()
{
  char status;
  double T,P,p0,a;
```

```
// You must first get a temperature measurement to perform a pressure reading.
  // Start a temperature measurement:
  // If request is successful, the number of ms to wait is returned.
  // If request is unsuccessful, 0 is returned.
status = pressure.startTemperature();
  if (status != 0)
  {
    // Wait for the measurement to complete:
    delay(status);
    // Retrieve the completed temperature measurement:
    // Note that the measurement is stored in the variable T.
    // Use '&T' to provide the address of T to the function.
    // Function returns 1 if successful, 0 if failure.
    status = pressure.getTemperature(T);
    if (status != 0)
    {
      // Start a pressure measurement:
      // The parameter is the oversampling setting, from 0 to 3 (highest res, longest wait).
      // If request is successful, the number of ms to wait is returned.
      // If request is unsuccessful, 0 is returned.
      status = pressure.startPressure(3);
      if (status != 0)
      {
        // Wait for the measurement to complete:
        delay(status);
                // Retrieve the completed pressure measurement:
        // Note that the measurement is stored in the variable P.
        // Use '&P' to provide the address of P.
        // Note also that the function requires the previous temperature measurement (T).
        // (If temperature is stable, you can do one temperature measurement for a number of pressure
measurements.)
        // Function returns 1 if successful, 0 if failure.
status = pressure.getPressure(P,T);
        if (status != 0)
        {

        return(P);
        }
        else Serial.println("error retrieving pressure measurement\n");
}
      else Serial.println("error starting pressure measurement\n");
    }
    else Serial.println("error retrieving temperature measurement\n");
  }
  else Serial.println("error starting temperature measurement\n");
}
```