

UNIVERSITY OF VAASA

FACULTY OF TECHNOLOGY

TELECOMMUNICATION ENGINEERING

Antonios-Fanourios Plytas

**DESIGN AND IMPLEMENTATION OF A DATA COLLECTION
SYSTEM WITH RASPBERRY PI FOR GREENHOUSE
MONITORING**

Master's thesis for the degree of Master of Science in Technology submitted for inspection,
Vaasa, 13 June, 2014

Supervisor

Professor Timo Mantere

Instructor

M.Sc. (Tech) Tobias Glocker

ACKNOWLEDGEMENT

First and foremost I would like to express my very great appreciation to my supervisor, Professor Timo Mantere. I am particularly grateful for the assistance and valuable suggestions given by my instructor Tobias Glocker. Without his assistance this Master's thesis would not be completed.

I would like also to those people who share their knowledge with me during these years.

Thanks to my family for encouraging me and support me emotionally and financially.

TABLE OF CONTENTS

ABBREVIATIONS	5
ABSTRACT	7
1. INTRODUCTION	8
2. BACKGROUND AND THEORY	10
2.1. The Principles of Greenhouse.....	10
2.2. Humidity and Temperature.....	11
2.3. SQLite Database	12
2.3.1. SQLite Features.....	13
2.3.2. Limitations	18
2.4. Raspberry PI	19
2.4.1. Hardware Specifications	20
2.4.2. Raspberry Pi Distributions	23
2.5. Arduino Uno	24
2.5.1. Hardware	24
2.5.2. Software	26
2.6. Serial Communication Interfaces	26
2.6.1. Universal Asynchronous Receiver/Transmitter (UART).....	27
2.6.2. Single Wire.....	28
3. HARDWARE IMPLEMENTATION	33
3.1. System Overview.....	33
3.2. DHT11 Sensor	34
3.2.1. Overview	34
3.2.2. Communication Process.....	36
3.3. L293D Half-H Motor Driver.....	37
3.3.1. H-Bridge basics.....	37

3.3.2. Introduction to L293D IC.....	39
3.4. Design and Implementation of L293D board.....	40
4. SOFTWARE IMPLEMENTATION	43
4.1. Setting up the Raspberry Pi	43
4.2. Software Design of the Raspberry Pi	48
4.3. Software Design of the Arduino Uno	50
4.4. Developing a Dynamic Website	53
4.4.1. Understanding the Website Development.....	53
4.4.2. Website Design	54
5. SYSTEM TESTING	58
6. EXPERIMENTS	62
7. CONCLUSION & FUTURE WORK	64
REFERENCES	65
APPENDIXES	74
APPENDIX 1	74
APPENDIX 2.....	75
APPENDIX 3.....	78
APPENDIX 4.....	83

ABBREVIATIONS

AC	Alternative Current
ACID	Atomicity, Consistency, Isolation, Durability
CEC	Consumer Electronics Control
CPU	Central Processing Unit
CSS	Cascading Style Sheet
DC	Direct Current
DHCP	Dynamic Host Configuration Protocol
DIP	Dual In-line Package
EEPROM	Electrical Erasable Programmable Read-Only Memory
EPS	External Power Supply
GB	Giga Byte
GND	Ground
GPIO	General Purpose Input Output
GUI	Graphic User Interface
HD	High Definition
HDMI	High-Definition Multimedia Interface
HTML	Hypertext Markup Language
HTTP	Hyper Text Transfer Protocol
I ² C	Inter- Integrated Circuit
IDE	Integrated Development Environment
IP	Internet Protocol
KB	Kilo Byte
LAMP	Linux, Apache, MySQL, PHP , Perl server

LSB	Least Significant Bit
MB	Mega Byte
MCU	Micro Controller Unit
NTC	Negative Temperature Coefficient
PoP	Package on Package
PWM	Pulse Width Modulation
RDBMS	Relational Database Management System
ROM	Read-Only Memory
SDRAM	Synchronous Dynamic Random-Access Memory
SoC	System on Chip
SPI	Serial Peripheral Interface
SQL	Structured Query Language
SRAM	Static Random-Access Memory
SSH	Secure Shell
TTL	Transistor-Transistor Logic
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
URL	Uniform Resource Locator
WiFi	Wireless Fidelity

UNIVERSITY OF VAASA**Faculty of technology**

Author:	Antonios-Fanourios Plytas
Topic of the Thesis:	Design and Implementation of a data collection system with Raspberry Pi for Greenhouse Monitoring.
Supervisor:	Professor Timo Mantere
Instructor:	Tobias Glocker
Degree:	Master of Science in Technology
Degree Programme:	Degree Programme in Information Technology
Major of Subject:	Telecommunication Engineering
Year of Entering the University:	2012
Year of Completing the Thesis:	2014

Pages: 83

ABSTRACT:

The need to produce high quality vegetables or flowers throughout the whole year, led to the development of automated greenhouses. Automation systems contribute to achieve the optimal conditions indoor, enable to be monitored and controlled from distance. Measurements such as humidity and temperature are critical for plantation growth thus they need to be accurately monitored and controlled.

The aim of the thesis is the design and implementation of a data collection system in a greenhouse using two most popular programmable boards, Raspberry Pi and Arduino Uno. The system collects data of the most essential parameters, as humidity and temperature, which contribute to the desired climate. To control temperature and humidity, a fully automated ventilation system maintains the greenhouse's interior to optimal conditions. The humidity and temperature measurements are stored in a SQLite database, which is installed on the Raspberry PI. Over a dynamic webpage the data can be retrieved from almost every place in the world.

KEYWORDS: Greenhouse, Raspberry Pi, Arduino Uno, SQLite.

1. INTRODUCTION

The reproduction control of plants out of season, under modifying conditions is the major purpose of a greenhouse (Sammons 2005). The most essential aspects of plant growth are the level of humidity and temperature, providing the required improvement of productivity and the quality of products (Castilla 2013). Those features can be beneficial for the cultivation; however they can be harmful for the plants into an uncontrolled interior place.

Over the few decades, automated control systems have been developed for greenhouses, contributed to improve the quality and quantity of production by constant monitoring and controlling the cultivation from a long distance.

The aim of the thesis is the design of a monitoring system for a greenhouse. The system collects information from the inside of the greenhouse such as air temperature, relative humidity and dew point which are vital factors for plant growth. The two basic components of the system are the credit-card sized board, Raspberry Pi, which provides server services and Arduino Uno which provides a powerful I/O interface.

A DHT11 temperature and humidity sensor collects data that are sent to the Arduino board. To maintain the desired humidity levels into greenhouse, a ventilator involves replacing the inside air with fresh air. A DC motor operates as ventilator according to humidity levels. To drive this motor, the L293D H-bridge motor driver IC is used by the system and it is capable to drive DC motors in two directions under the control of a microcontroller. In Raspberry Pi, a python script initiates the communication with the Arduino Uno board, sending a data request.

Once the data are received by the Raspberry Pi, they are stored in a SQLite database. Saving the data to database, the user can retrieve them by linking to them in a web site. Thus the user has the real time monitoring of a greenhouse.

The thesis starts with the fundamentals of a greenhouse until the implementation of system.

The *second chapter* begins with providing the essential information about the function of greenhouse and the parameters that need to be controlled. The chapter continues introducing the basic components of the system, hardware and software and the connections of them. The purpose of this chapter is to familiarize the reader with the basic elements of the system and to give some background context.

The *third chapter* initializes the system's implementation, exploring the hardware part. This chapter is focused on system design, connections between controllers and sensors.

In *chapter four* the reader has the opportunity to see the correlation of different programming languages and how to provide the interface between the hardware and the software.

The *chapter five* is the system testing in order to confirm that the hardware and software work properly and identify potential problems.

Chapter six is the experimental part and is focused on efficiency of the system such as power consumption.

In the conclusion, the *final chapter* summarizes the concept of the thesis and also the future work.

2. BACKGROUND AND THEORY

2.1. The Principles of Greenhouse

Prior to the description of the system and its key elements, the thesis will focus on necessary principles of greenhouse for understanding operating procedures. Bakker (2001:1) describes the greenhouse as “barrier” which isolates the cultivation from the external environment creating optimal conditions for the plants and protecting them, from the external factors which could harm the cultivation. More precisely, a greenhouse is a structure covered by transparent material such as glass or plastic. The plastic material allows the short-wavelength infrared light to enter inside but it blocks the long-wavelength light. The radiation absorbed by the plants and soil, increasing the temperature. The long wavelengths of radiation which emitted by the heated objects are blocked by the transparent material, rising the interior temperature. This process is known as *greenhouse effect* (see Figure 1.). The air close to earth becomes warmer than higher therefore the phenomenon of convection occurs. Because of the trapped radiation, the warm air becomes warmer each time it rises and falls. The heat leads also to water evaporation which increases the humidity level, creating desirable conditions for the plants. (Nave 2012a, Nave 2012b, Bowling 1987).

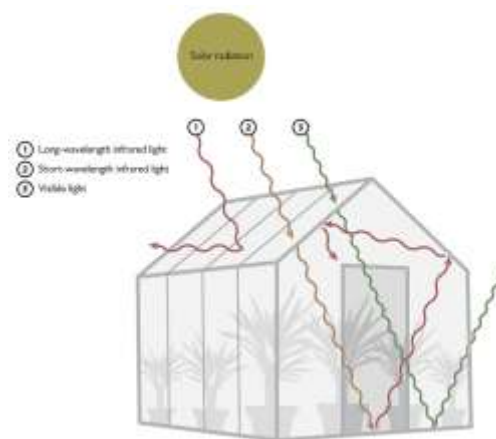


Figure 1. Greenhouse effect (Littmark 2013).

2.2. Humidity and Temperature

To achieve desirable conditions inside the greenhouse it is necessary to control two of the major factors in the plant growth. These factors are humidity and temperature which are interrelated. Humidity defines the amount of moisture in the atmosphere. Relative Humidity (RH) is the most commonly used measurement of humidity and it is expressed as “ the ratio of the amount of moisture in the air at a specific temperature to the maximum amount that the air could hold at that temperature, expressed as a percentage” (McColl 2002). The relative humidity is calculated by using the following equation:

$$\text{Relative Humidity}(RH) = \frac{\text{Actual Vapor Density}}{\text{Saturation Vapor Density}} \times 100 \quad (1)$$

Both actual vapor density and saturation vapor density are measured in grams per cubic meter (gm/m^3). For example, if the air temperature is 32°C , the saturated vapor density is $33.97 \text{ gm}/\text{m}^3$. In case that the actual humidity is $18.91 \text{ gm}/\text{m}^3$ at the same air temperature then the relative humidity is 55.65% (Nave 2012c). When the relative humidity reaches 100%, the air is saturated and this point is known as dew point (Nave C.R. 2012d).

The dew point is a measurement of humidity which is related with air temperature and relative humidity. More precisely, it is the temperature at which the relative humidity rise up to 100% while the temperature is cooled with constant pressure (Nave C.R. 2012d). Saturated air will release some moisture through forms of precipitation (Wikipedia 2014a) or condensation (Wikipedia 2014a), if the temperature decreased more. For example, when the relative humidity is 85% and the temperature is 20°C , condensation occurs at 17.3°C . At 95% of humidity with the same air temperature, the condensation occurs at 19.2°C .

High humidity concentrations cause plant diseases as well as reduction of transpiration. Eshenaur (2004:1-3) notes that “the relative humidity is usually 25%-70% during the day in greenhouses”. In order to control the humidity to these levels there are several techniques improving the efficiency of cultivation. A ventilation system is very important for the reduction of temperature and humidity as well as air circulation. The system uses ventilators for the inflow and outflow of air diffusing it along the greenhouse. (Eshenaur 2004, Both 2008: 1-2.)

2.3. SQLite Database

The first thing that comes to mind when hearing of SQL databases is Microsoft’s SQL, Oracle database etc. Nowadays as the range of embedded systems increases, the needs for fast, lightweight and reliable relational database management systems (RDBMS) (Kreibich 2010:1) increase as well. The SQLite is designed for this purpose. More specific, according to the official website “SQLite is an in process library that implements a self-contained, serverless, zero-configuration, transactional SQL data engine” (SQLite 2014a). In other words, SQLite database is a single file located in the disk which can be easily shared or copied to other media. It is designed without external server which means zero-configuration and simple operation. Furthermore, depending on the target platform and optimization settings, SQLite library is less than 350KB in size on x86 and less than 400KB in size on x64, making suitable for embedded systems (SQLite 2014b). In addition, SQLite is in the public domain which means that the source code is not under license or copyright law, thus it is free to be modified, copied, distributed and used it for any purpose (SQLite 2014c).

As other popular RDBM systems, the SQLite is used to achieve and manage the data in multiple tables however some features make it exceptional.

2.3.1. SQLite Features

Despite the size, SQLite provides significant features making it exceptional for the embedded systems.

The SQLite is self-contained which means that the entire database is contained into a less than half of megabyte library requiring the minimal resources that makes it suitable to embedded systems. In other words, a single disk file contains the entire database in which can be backed up into USB memory stick or shared by email. (Kreibich 2010: 2.)

In contrast with the most popular RDBM systems, the SQLite runs directly on the disk without any administration control or other complex configurations, which make it simple and convenient for portable environments. Figure 2 illustrates the SQLite architecture in which every program communicates directly with SQLite file without intermediate process. In case of traditional SQL databases, the process between client and server is called inter-process communication. In that case the client sends request to the server and the server responds with the selected data retrieved from the database. This procedure increases the overhead and administration complexity thus it is improper for embedded systems. (Kreibich 2010: 2-4.)

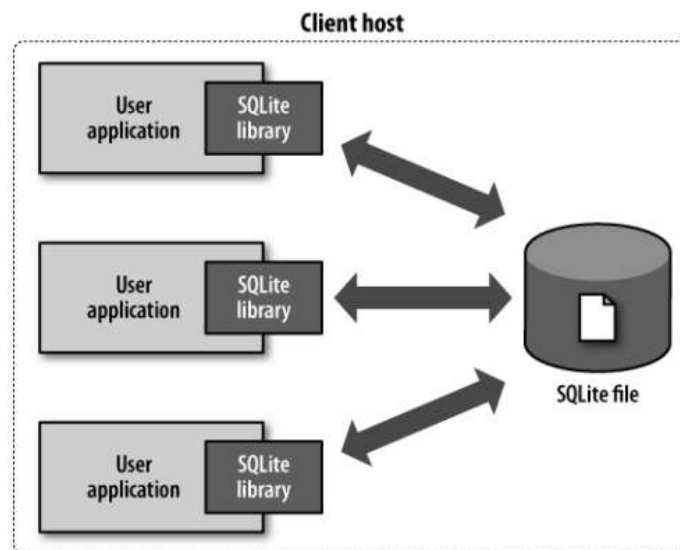


Figure 2. User-SQLite Database interaction (Kreibich 2010: 3).

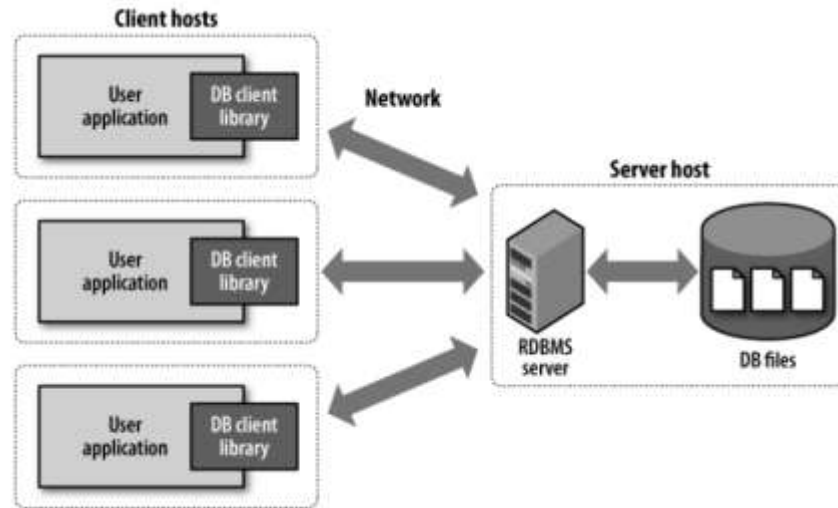


Figure 3. User-Server host interaction (Kreibich 2010: 3).

By eliminating the external dependencies, SQLite has no need of administrative support or installation process. SQLite runs independently without need to recover after a system glitch.

Another essential advantage is the dedication of code and documentation to the public domain. This means that SQLite code and documentation is distributed free without copyright licenses thus anyone is free to copy, modify or publish the source code without any restrictions. (SQLite 2014c.)

Furthermore, another feature that makes it essential for the embedded system is the small library which is less than 400KB and requires less than 256 KB of memory without optional features (SQLite 2014b).

The integrity of data in RDBM systems like SQLite is defined by the atomic transactions in which a series of commands either all executed or not at all. In SQLite the transaction remains atomic even if a system crashes or power loss occurs during the process. The atomicity however is a part of transaction properties known by the acronym ACID (SQLite 2014d).

Every transaction should have *atomic* characteristics which ensure that all transactions are committed or aborted.

The transaction is *consistent* when the database transit from accurate state to another following integrity rules (Kreibich 2010: 52).

An *isolated* transaction is referred also as serializable which allows the execution of transactions in sequence without affecting each other. This implies that the transaction should include the previous properties that have already been mentioned. (Bernstein and Newcomer 2009: 13-14.)

Durability means the nonvolatile storage of records when the transaction is committed even if power loss or system crash occurs. (Bernstein and Newcomer 2009: 14-15.)

SQLite uses five different lock transactions to support the serializable execution of transactions. These lock states (see Figure 4) ensure that applications have the permission to read or write.

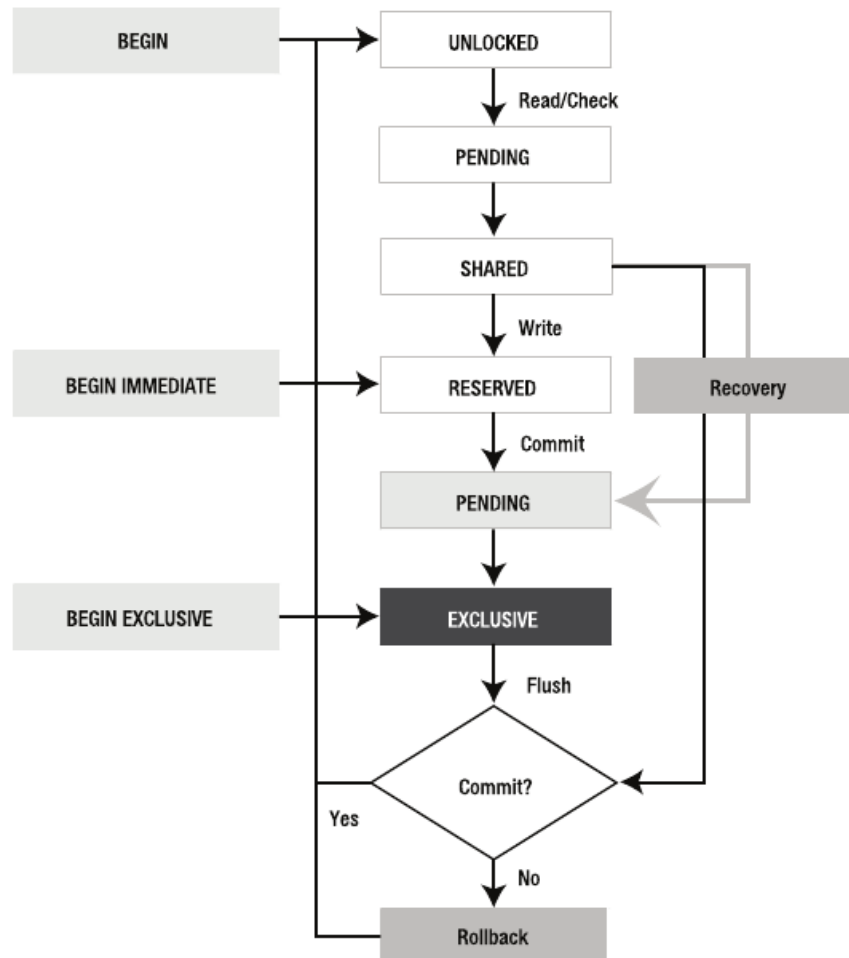


Figure 4. SQLite transactions (Grant 2010: 140).

UNLOCKED: This is the default state without any lock on the database. No one can read or write on the database at this state.

SHARED: The Shared lock gives the permission to multiple readers at the same time but not writers.

RESERVED: As the name implies, this lock reserves the database for future modification but it is still reading. This means that the reserved lock and shared lock have almost the same characteristics and can coexist but only one reserved lock at the time.

PENDING: The Pending lock is the last step before modification. It allows the readers to keep reading but block the new Shared locks. When the current Shared locks are released, then the database can be modified, getting an Exclusive Lock.

EXCLUSIVE: Finally, the Exclusive Lock gives the permission for updating the data allowing only itself and no other kind of locks. (SQLite 2014e.)

To begin with, the Unlock state is the default state where no one can read or write. The Pending lock after Unlock state represents the path from unlocked state to Shared lock. Prior SQLite can write to a database, it must read the database to check its content. At this step the database obtains a Shared lock which gives the permission for reading to multiple readers but not writers.

To *begin immediate* transaction obtains a Reserved lock which reserves the writer for a future update of data. To prevent the loss of data in case of power failure or system crash, SQLite creates a file before any alternation of data. This file is called rollback journal and contains a copy of original contents of database, therefore it can recover the database back to its original state. The prior step of Exclusive lock is the Pending lock which gives permission to the current Shared locks to keep reading but deny the access to new Shared locks.

The next step is the Exclusive lock which guarantees that only one writer can update the data and no other kind of lock is active. In that case a flush must occur, putting the new data which are written into permanent storage, improving at the same time the integrity of database. If the data are not committed properly then the rollback activates and restores the database back to its original size. (SQLite 2014f, Grant and Owens 2010: 138-142.)

2.3.2. Limitations

The simplicity is the elemental characteristic of SQLite. It is fast when it is not facing complex queries, small because it runs into a simple disk file and reliable as a result of simplicity.

As already mentioned, the SQLite is designed for embedded systems therefore there are some limitations when it is used for a different purpose. Unlike the large-scale databases, SQLite operates faster when the user searches or retrieves data. However, it does not work the same way on larger databases when the complexity increases. (Grand et al 2010: 11.)

SQLite has the ability to permit multiple readers simultaneously, however only one has the permission to write at a time and only for a few milliseconds. According to the official SQLite website, (SQLite 2014f) in order to prevent the database update while multiple readers read the database simultaneously, a shared lock mode obtained. On the other hand, another lock mode enables before any modification on database. When reserved mode obtained, SQLite gives the permission to a single process of modifying the database at a time while multiple processes can read the data. Consequently, SQLite is unsuitable for applications with high concurrency.

Another limitation is when the database is shared with multiple computers over a network file system, increasing the delay as a result of reduction the database performance. In addition, many glitches are contained on network file systems causing volatility to locking logic. In that case the locking mechanism breaks, allowing more than one process to update the database at the same time. This leads to the database corruption. (SQLite 2014f, Grant et al 2010: 12.)

2.4. Raspberry PI

The Raspberry PI is a tiny, inexpensive, Linux computer capable to support different kind of peripherals. The primary aim of the project according to the co-founder, Eden Upton, is to provide an inexpensive computer which will encourage the new generation to learn programming beyond the closed confines of universities while creating the future engineers for the industry (Halfacree 2012, Raspberry Pi Foundation 2014a). The Raspberry foundation released two models, model A and B (see Figure 5). Both models look similar, however there are differences on memory capacity, cost and connectivity. The 35\$ model B supports 512MB SDRAM while the 25\$ model A supports 256MB. In addition, on model B has dual USB 2.0 connector while model A has only one. Moreover, the model A lacks a RJ45 Ethernet port. (Raspberry Pi Foundation 2014b.)



Figure 5. Raspberry Pi Model A and Model B (Monk 2010:2).

2.4.1. Hardware Specifications

The main component of Raspberry Pi is the processor which is based on mobile device chipset. The ARM1176JZF-S which is a part of the Broadcom BCM2835 SoC, is a high efficient 32bit CPU based on ARMv6 architecture and clocked at 700MHz. (Sjogelid 2013: 8.)

The ARM11 (ARM Ltd) micro architecture provides high performance in low cost and low power consumption, making it suitable for a wide range of mobile and embedded applications. One of the beneficial features is the 8-stage pipeline structure, which obtains higher throughput compared to previous cores and low latency due to effective optimization techniques. The memory management is another feature of ARM11 which increases the performance reducing the latency of data and providing separate data and instruction caches. (Brash 2002: 3-4, Cormie 2002: 1-9). Concerning the graphics, the BCM2835 contains the VideoCore IV multimedia co-processor, providing high level video and multimedia performance with 1080p HD video and 3D graphics (Broadcom 2014).

The 512MB/ 256MB RAM of the model B and A respectively is a part of SoC chip. Memory and CPU are designed vertically using the method of Package on Package (PoP). This method saves space on board, the stacking the memory on the top and the CPU package on the bottom of package. (Horan 2013: 13.)

Instead of hard disk, the Raspberry Pi supports an SD card for data storage. Concerning the range of capacity there is variety of storage sizes, from 4GB at least up to 32GB which is the recommended range for the Raspberry Pi (Sjogelid2013: 11).

As it already mentioned, the model B provides dual USB 2.0 which enables connection with peripherals such as keyboard, mouse, WiFi dongles etc. To extend the system, a powered USB hub is suggested, considering that the maximum recommended current is 100mA. (Horan 2013: 2-3.)

Ethernet port allows the communication between Raspberry Pi and network through a cat 5 twisted-pair cable. Both USB 2.0 and Ethernet are controlled by the LAN9512 (see Figure 6) which contains a Hi-Speed USB 2.0 hub and an Ethernet controller in one chip. The following block diagram presents that both USB hub and Ethernet controller are interconnected in order to provide network functions sharing the only one upstream USB port. (Horan 2013: 2-3, Microchip Technology 2012: 1-8.)

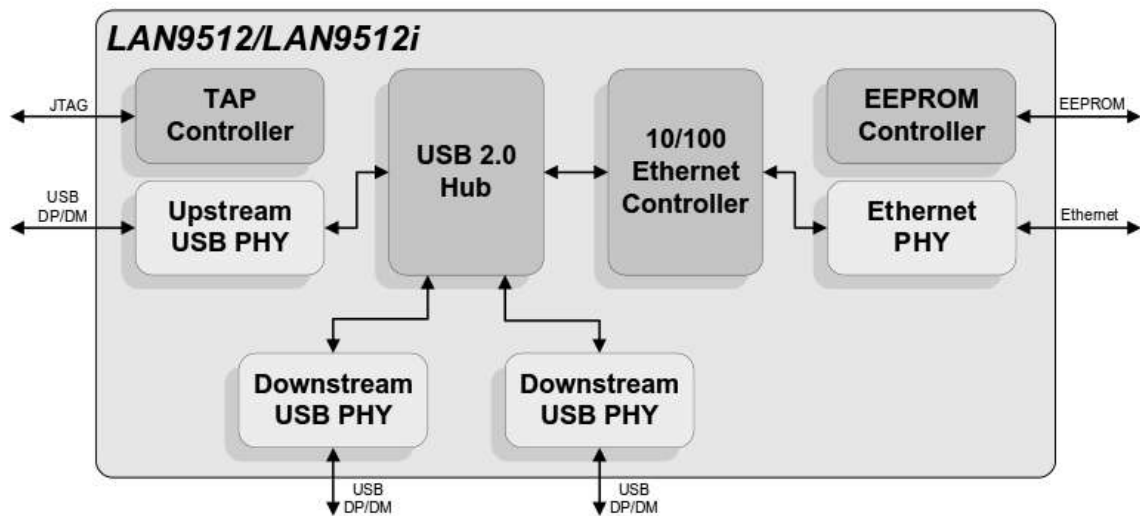


Figure 6. USB 2.0 and 10/100 Ethernet Controller (Microchip Technology 2012).

The High-Definition Multimedia Interface or HDMI standard provides connection with high definition signals up to 1920 x 1200 pixels and eight channels of uncompressed audio. One beneficial feature of HDMI is the Consumer Electronics Control (CEC) (HDMI 2014b) which allows the Raspberry Pi to be fully controllable using the TV's remote control. (Sjogelid 2013:10, HDMI 2014a.)

Raspberry Pi provides also an analog video transmission via the composite video output, allowing to use the television as monitor (Dennis 2013: 10).

For analog audio, the Raspberry Pi supports the 3.5mm analog audio jack enabling the connection of headphones and speakers (Dennis 2013:10).

The General Purpose Input/Output (GPIO) is a low level interface composed of 26-pin header. This feature allows peripherals and expansion boards to interact directly with processor. The GPIO supports serial communication interfaces and also supply of 5V, 3.3V and ground. (Elinux 2014.)

The 5V MicroUSB has been chosen in order to supply the Raspberry Pi with 700mA. MicroUSB is a common External Power Supply (EPS) thus it is also common with mobile charges. (Upton 2011, CENELEC 2011.)

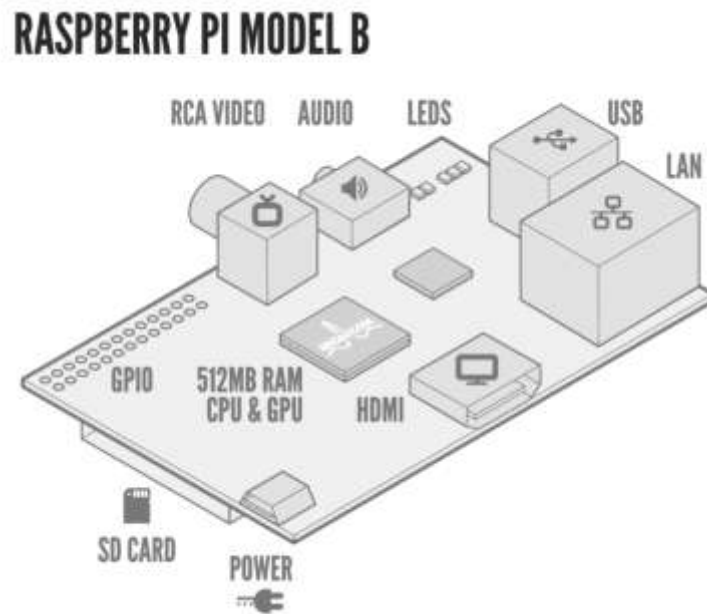


Figure 7. Model B features. (Raspberry PI Foundation 2014).

Table 1. Specifications of Raspberry Pi Model B (Elinux 2014b).

SoC :	Broadcom BCM2835
CPU :	700MHz ARM1176jZF-S
GPU :	Broadcom VideoCore IV
Memory :	512MB
USB 2.0 :	2 Ports
Video outputs :	Composite RCA, HDMI
Audio outputs :	3.5mm jack, HDMI
Storage :	SD card
Network :	10/100 wired Ethernet RJ45
Low-Level peripherals :	GPIO pins, SPI, I2C, UART
Power rating :	700mA
Power source :	5V

2.4.2. Raspberry Pi Distributions

As microcomputer, Raspberry Pi runs Linux operating system. There are plenty of Linux distributions compatible with desktop computers, however they are not appropriate for the mobile device chip of Raspberry Pi. The designers recommend specific distribution such as Raspbian, (Raspbian 2014) which is based on Debian Wheezy armhf (Raspberry Pi Foundation 2014c).

Unless Raspbian, there are the following distributions:

- Pidora which is a mix of Fedora operating system with third-party software (Red Hat 2014, Raspberry Pi Foundation 2014c).
- Arch Linux which is built for ARM processors (Arch Linux ARM 2014, Raspberry Pi Foundation 2014c).

2.5. Arduino Uno

Arduino is described by Massimo Banzi (2011: 1) as “an open source physical computing platform based on a simple input/output (I/O) board and a development environment that implements the Processing language.”

The inexpensive, printed circuit board comes with Atmel’s microcontrollers, capable to develop interactive prototypes. In other words, the various capabilities of I/O interface allow interaction with a wide range of sensors or control lights and motors, through the code. More than fifteen official prototype boards have been developed. (Arduino 2014a.)



Figure 8. Arduino Uno board (Arduino 2014).

2.5.1. Hardware

The Arduino Uno (see Figure 8) board is composed of the high performance and low consumption ATmega328P-PU, (Atmel Corporation 2014a) designed with picoPower Technology (Atmel Corporation 2014b). The 28-pin DIP package runs at 16MHz offering 32KB flash memory for storage. It provides various features such as, fourteen digital channels and six analog, six PWM channels, programmable serial USART, SPI and I²C interfaces, everything in a single package. (Arduino 2014b.)

The Arduino Uno can be powered in several ways. The common way is using a USB cable connected to a personal computer providing 5V. When it operates independently, it can be powered via AC/DC adaptor connected to a modular barrel connector, providing voltages between 7V to 12V. Arduino Uno can be operated also the connecting a battery or series of batteries to the DC power connector or using the input voltage of I/O interface. (Arduino 2014b, Wheat 2011: 5.)

The board has strong capabilities of I/O interface providing 27 pin connectors. A set of 14 pins can be used as digital I/O, with six of them provide 8-bit PWM, four pins support SPI communications, two serial pins provide UART TTL serial communication and two external interrupts. On the other side of the board, analog inputs and supply voltages complete the grouping of the I/O pins. (Arduino 2014b.)

On the following table the technical features are specified:

Table 2. Arduino Uno specifications (Arduino 2014).

Name	Arduino Uno
Microcontroller	ATmega328P-PU
Operating Voltage	5V
Input Voltage	7-12V
Digital I/O Pins	14
PWM	6
Analog Input Pins	6
DC Current per I/O Pin	40mA
DC Current for 3.3V Pin	50mA
Flash Memory	32KB
SRAM	2KB
EEPROM	1KB
Clock Speed	16MHz

2.5.2. Software

Arduino Uno in order to become programmable uses the Arduino Integrated Development Environment (IDE) cross-platform software. It is a Java application based on Processing, allowing the code to be written in simplified C/C++, known as Arduino programming language (Arduino 2014c, McRoberts 2010: 4). The written code on IDE can be easily compiled and uploaded to microcontroller through the port. Arduino uses a small pre-installed code which reduces the complexity of loading code in microcontroller, known as boot loader (Arduino 2014b). In addition, to improve the function of program, Arduino provides a collection of libraries for different kind of purpose such as the Wire library which enables interaction between devices or the Servo library which allows the control of servo motors. Each library is available for everyone to download and reuse them in different projects. (Arduino 2014d.)

2.6. Serial Communication Interfaces

The fundamental idea of serial communication is the interaction between digital systems which share the same communication protocol and exchange information in series, one bit at a time. In addition, serial interfaces transmit their data using a single wire, reducing the need of extra hardware which is the main advantage over the parallel interfaces. The serial communication interfaces are classified into two main groups: synchronous and asynchronous serial interfaces. In the synchronous serial communication, the data and therefore all the devices are synchronized with the common clock. For that purpose interfaces require two lines, one for data and one for the clock. On the other hand, in asynchronous serial communication, external clock is not required thus only one line is needed. Instead of a clock, asynchronous communication defines parameters which are common for both master and slave devices. (Summerville 2009: 77, SparkFun Electronics 2012). Afterwards two asynchronous protocols are described, UART and 1-Wire which are essential for the interaction between embedded systems and sensors.

2.6.1. Universal Asynchronous Receiver/Transmitter (UART)

The UART (see Figure 9) is a protocol providing an asynchronous serial communication over a single wire, between two systems. Compared with a synchronous serial communication, the UART does not need synchronization from external clock signal (Fernandez, Darig 2009, Silicon Labs 2014). However, it relies on the internal configurations for both systems. At the transmitter side, a parallel to serial conversion occurs and the data transmits sequentially bit by bit. On the other hand, the receiver reassembles the bits and converts them from serial to parallel. The communication begins adding first a start bit in order to inform the receiver that a sequence of bits is ready to arrive. The receiver starts to accept a frame of 8 bits with the Least Significant Bit (LSB), according to baud rate. To insure the data integrity, parity bit may be sent, which must be agreed from both transmitter and receiver. Eventually, a stop bit is sent, indicating a new frame can begin after this bit. (Shibu 2009: 48-49, Lipovski 1999: 408-409, Silicon Labs 2014: 7-8.)

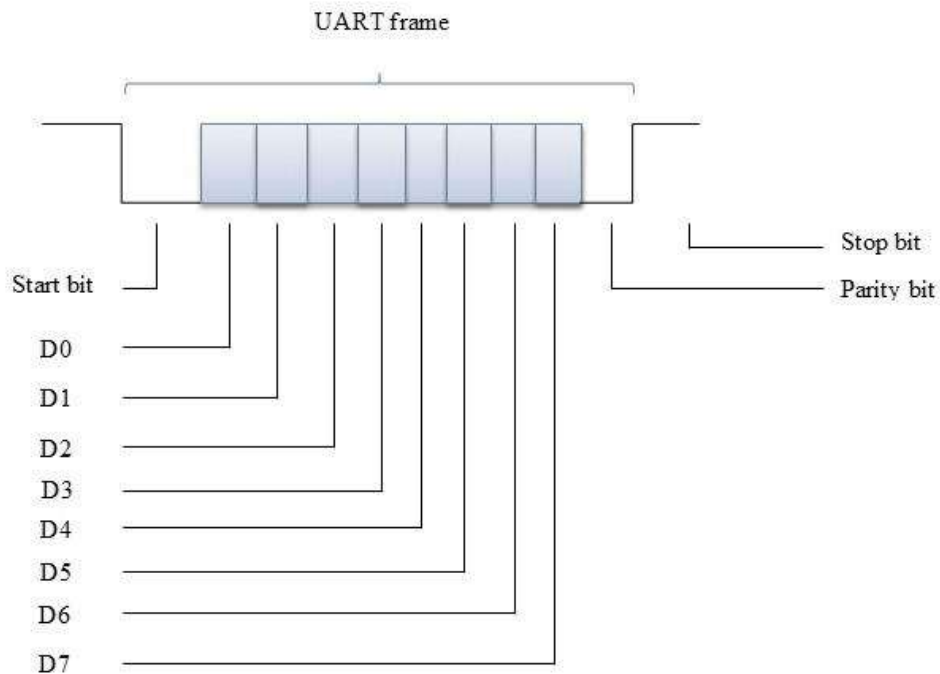


Figure 9. UART frame.

2.6.2. Single Wire

The Maxim single wire bus is a half-duplex, bidirectional protocol that transfers signals and power using only one wire between the master and the one or several slave devices (see Figure 10) reducing at the same time the interface complexity. The standard speed of 1-wire is approximately 15.4Kbps and increases up to 125Kbps using the overdrive mode. Master and slaves use open-drain ports pulled up to a supply through a 3V to 5V pull-up resistor, providing logic signals. In addition, the most of 1-wire devices are dependent since the power supply is provided by the host. (Linke 2008, Linke 2009, Maxim Presentation 2014, Willey 2001). The parasite-power devices comprehend the parasitic circuit which is composed by a capacitor and a diode in series. When the bus is in a high state the capacitor charges (see Figure 11) and when the low state occurs the capacitor provides energy to the internal oscillator and control circuit (see Figure 12). (Maxim Presentation 2014.)

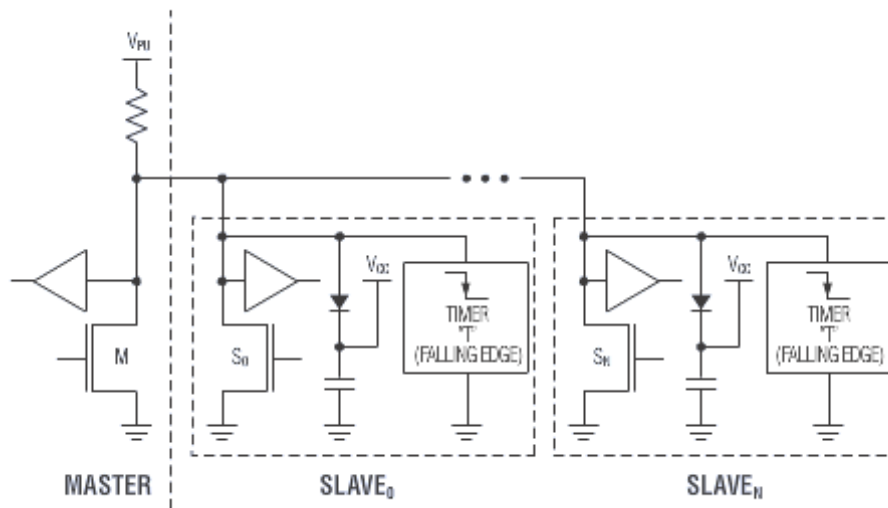


Figure 10. Master and Slave devices (Maxim Integrated 2014).

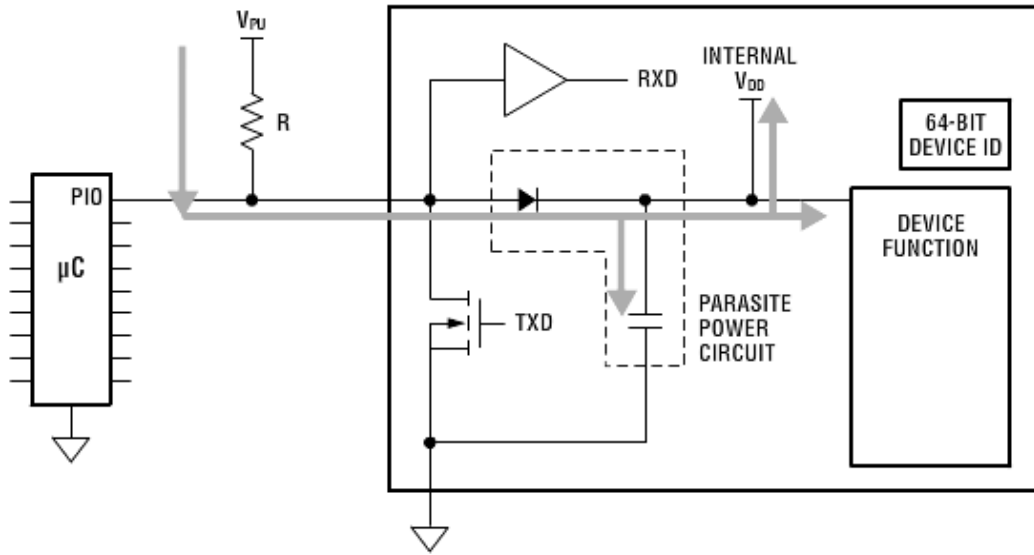


Figure 11. Parasitic power when the bus has high state (Maxim Integrated 2014).

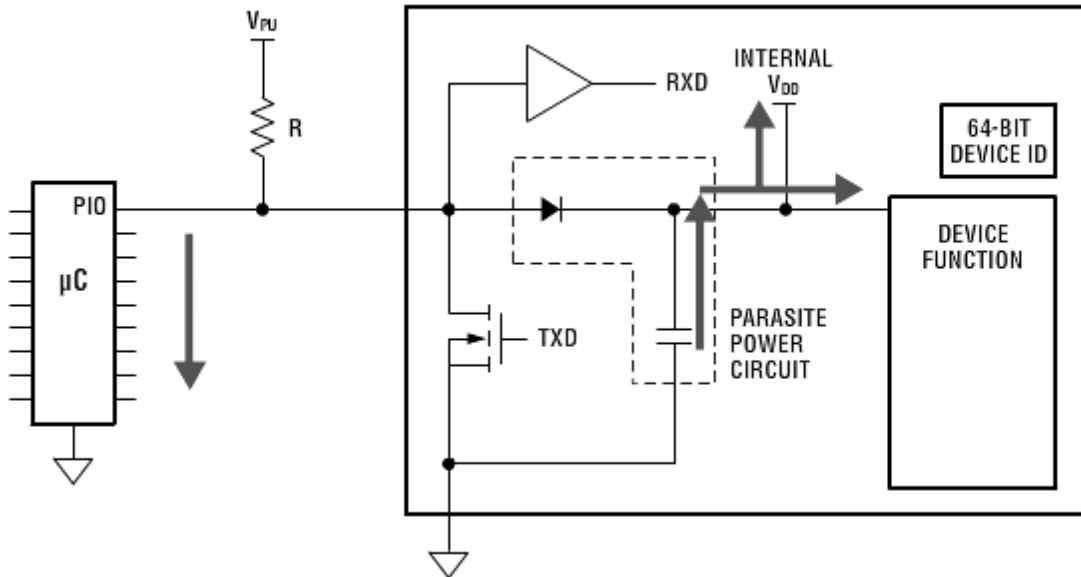


Figure 12. Parasitic power when the bus has low state (Maxim Integrated 2014).

Each slave device is identified by a unique 64bitROM number (see Figure 13) which addresses in every 1-wire device on the bus, where several devices are connected on it. The ROM number is comprised in three parts: an 8bit family code, a 48bit serial number and an 8bit CRC. (Shidu2009: 49-50, Maxim Presentation 2014.)

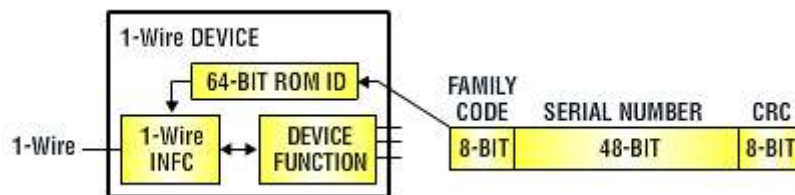


Figure 13. The ROM number (Maxim Integrated 2014).

The sequence of communication with a single wire device initiates with the line reset by the master. The slave device responds with a presence pulse. In the next step, the master sends a ROM command to the selected 1-Wire device initiating the communication. Finally, the Master is able to send a read/write function from or to the 1-Wire device. More precisely, the interaction between master and slave devices is divided into timeslots of 60μsec, sending a Reset pulse by the Master and holding the line low for >480μsec (see Figure 14). Afterwards, the Master releases the bus and the line pulling up to the high idle state for a period of 15 to 60μsec. All 1-wire devices on the bus respond with a Presence Pulse by pulling the bus low for a period of 60 to 240μsec. After this duration the line is released and the slave devices wait until the end of Recovery time when they return back to high state. (Shidu 2009: 49-50, Maxim Presentation 2014, Willey 2001.)

Figure 15 shows the procedure of read/write a logic 0 or 1. For writing logic 1 the master pulls the line low for 1 to 15μsec and releases it, allowing the line to return to the logic high state. This way the 1-wire devices identify the line status as a write 1 request. In case of

Write 0 the Host pulls the line low for a period of $60\mu\text{sec}$ maximum. Since the bus is low when the 1-wire device(s) samples the line, the slave devices identify this status as a Write 0. For reading a logic 1, the Master pull low the bus for a $15\mu\text{sec}$ and releases it for the rest of the time slot. On the other hand, in case of Read 0 the slave device keeps the line low for at least $15\mu\text{sec}$ and this is identified by the host as a Read 0. (Shidu pages 2009: 49-50, Maxim Presentation 2014, Willey 2001.)

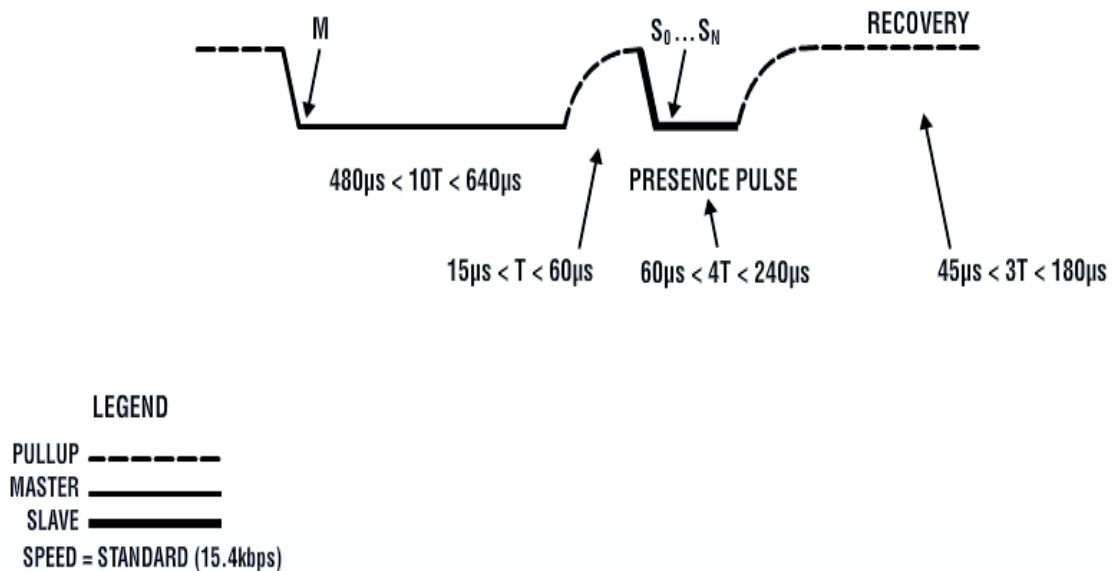


Figure 14. Reset and Presence pulse (Maxim Integrated 2014).

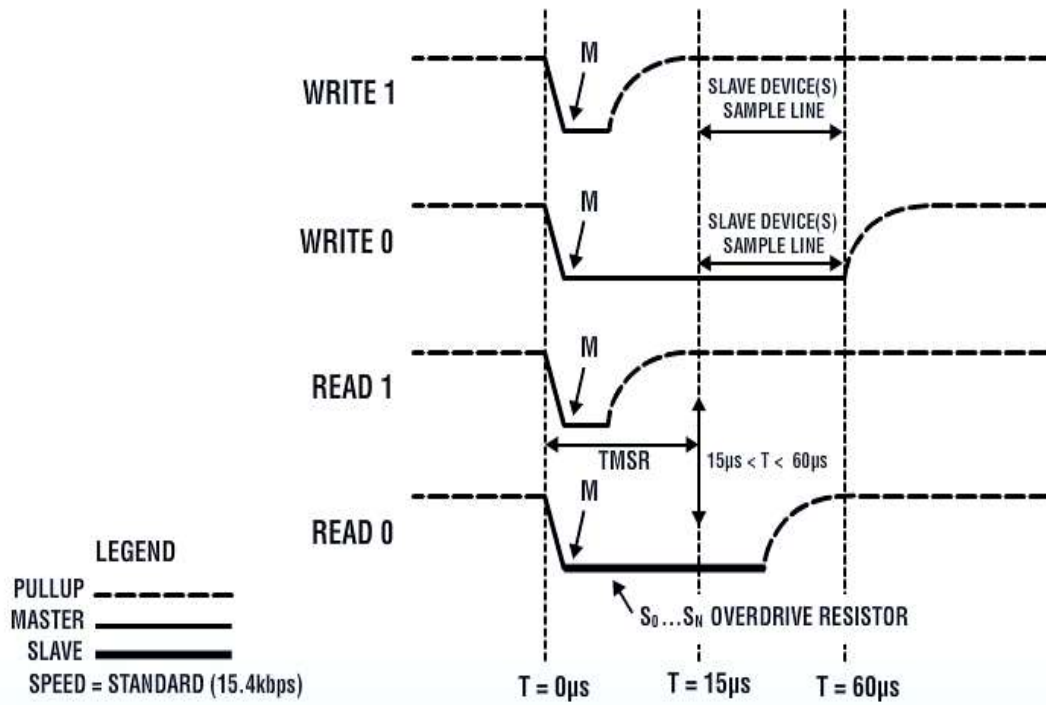


Figure 15. Read and Write waveform (Maxim Integrated 2014).

3. HARDWARE IMPLEMENTATION

3.1. System Overview

The system is designed in such a way that the user is able to receive the data through internet connection without any interference by the user. The following diagram illustrates, the five components of system:

- Raspberry Pi
- Arduino Uno
- DHT11 sensor
- L293D motor driver
- DC Motor

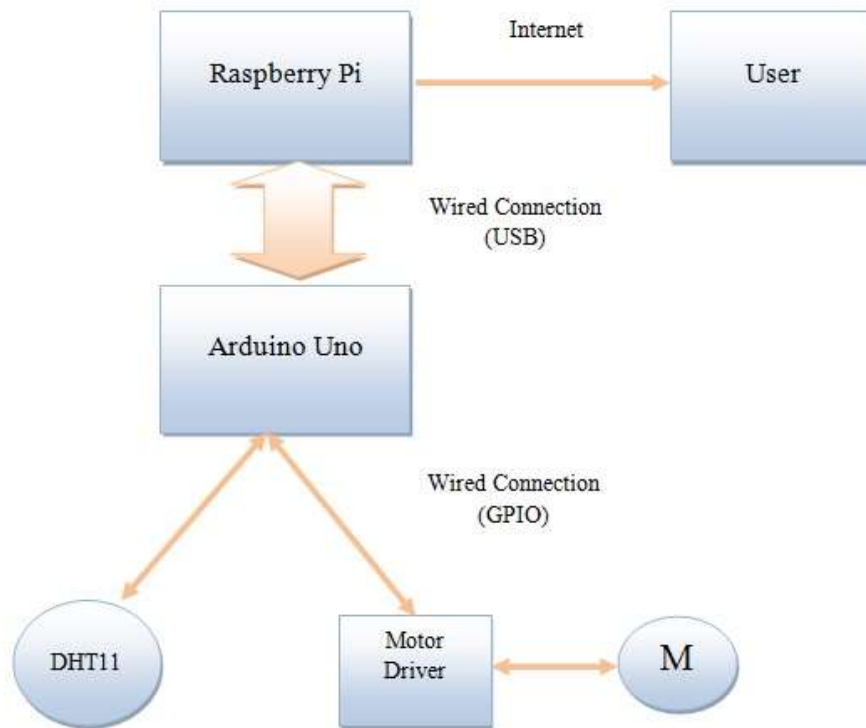


Figure 16. System overview.

The Raspberry Pi is connected to Arduino Uno through USB cable, which provides communication between the boards and power supply to Arduino. Only Raspberry Pi is connected to external power source. The DHT11 humidity and temperature sensor is connected to I/O interface of Arduino board, using the single-wire protocol. The L293D H-bridge motor driver is also connected to I/O interface of Arduino, using one of the six Pulse Width Modulation (PWM) outputs. The L293D IC is powered from the Arduino Uno with 5 Volts and from external power supply with 10Volts, in orders to provide enough power to motor.

3.2. DHT11 Sensor

3.2.1. Overview

The DHT11 humidity and temperature sensor is a single row 4-pin blue package which provides a calibrated digital signal output. This package includes a circuit which is composed of a 8-bit microcontroller, a resistive change-type humidity sensor and an NTC thermistor, providing high data integrity and performance. (D-Robotics 2010: 2-3.)

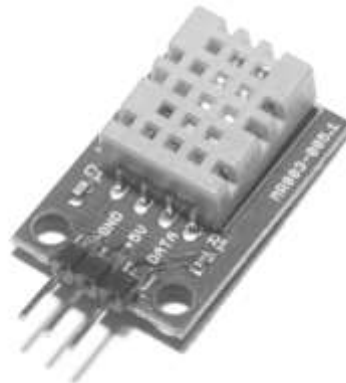


Figure 17. DHT11 humidity and temperature sensor.

The sensor operates at range from 3.5 to 5.5 V power supply. It offers a range of relative humidity measurements from 20% to 90% with $\pm 5\%$ and temperature measurements from 0°C to 50°C with $\pm 2^{\circ}\text{C}$. The detailed technical specification of DHT11 sensor are included (see Table 3). (D-Robotics 2010: 3-4.)

Table 3. Technical specifications of DHT11.

Parameters	Humidity	Temperature
Measurement range	20% - 90%	0°C - 50°C
Accuracy	$\pm 5\%$	$\pm 2^{\circ}\text{C}$
Repeatability	$\pm 1\%$	$\pm 1^{\circ}\text{C}$
Hysteresis	$\pm 1\%$	
Long-Term Stability	$\pm 1\%$ RH/year	
Response Time (sec)	6s-15s	6s-30s

In order to communicate with a microcomputer unit (MCU), DHT11 uses a single wire communication protocol. Figure 18 illustrates a typical connection between DHT11 sensor and MCU. The 1-wire protocol requires a 5K external pull-up resistor for a distance shorter than 20 meters. (D-Robotics 2010: 3-4.)

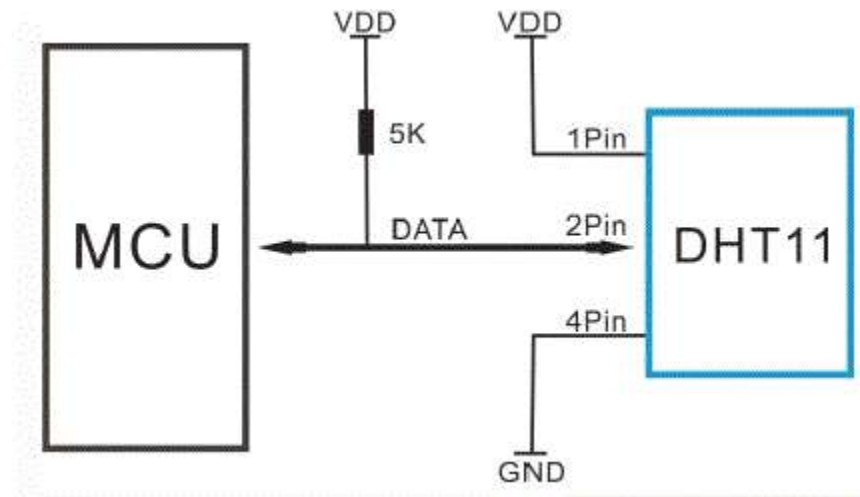


Figure 18. DHT11 connections (D-Robotics 2010).

3.2.2. Communication Process

The following diagram illustrates the communication process between MCU and DHT11 sensor. The process is initiated with the master's *start signal*, pulling the line low for at least $18\mu\text{s}$. After this time, the MCU releases the line and waits for sensor's response about $20\text{-}40\mu\text{s}$. The sensor which has detected the MCU's request, sends out a *response signal*, pulling the line low for $80\mu\text{s}$ and releases it. Once the MCU receives the response signal, it is ready to accept 40bits of data from DHT11. (D-Robotics 2010: 5-6.)

The data format is the following:

$$\underbrace{8 \text{ bit integral data} + 8 \text{ bit decimal data} + 8 \text{ bit integral data}}_{\text{Relative humidity}} + \underbrace{8 \text{ bit integral data} + 8 \text{ bit integral data} + 8 \text{ bit checksum}}_{\text{Temperature}}$$

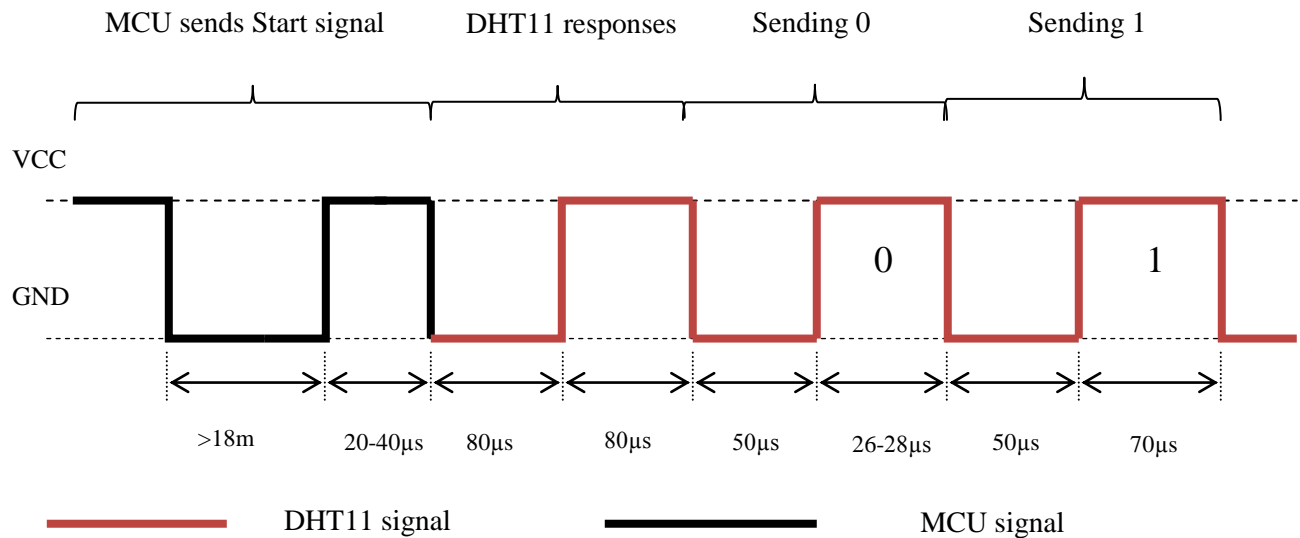


Figure 19. DHT11 waveform.

The DHT11, in order to send “0”, pulls down the line for 50µs and then it pulls in high voltage level for 26-28µs. In order to transmit “1”, pulls down the line for 50µs and raises it for 70µs. In other words, both 0 and 1 initiate with 50µs and the positive pulse provides the information. (D-Robotics 2010: 6-8.)

3.3. L293D Half-H Motor Driver

3.3.1. H-Bridge basics

The half-H bridge is composed of four switching components as the diagram indicates. Those switches open and close in pairs diagonally. In that way the bridge reverses the direction of current and thus it controls the direction of motor, changing the polarity. More precisely, when both switches SW1 and SW4 are opened (SW2 and SW3 are closed) then the current will be applied across the motor, thus the motor rotates clockwise. On the other

hand, when the switches SW2 and SW3 are opened (SW1 and SW4 are closed), then the current flows from the right side to the left one, rotating the motor anticlockwise. In case that SW1 and SW2 (or SW3 and SW4) are closed then the motor stops (McManis 2006). The following truth table summarizes the function of H-bridge.

Table 4. Truth table of H-Bridge.

SW1	SW2	SW3	SW4	Description
1(open)	0(close)	0(close)	1(open)	Turn right
0(close)	1(open)	1(open)	0(close)	Turn left
1(open)	1(open)	0(close)	0(close)	Stop
0(close)	0(close)	1(open)	1(open)	Stop

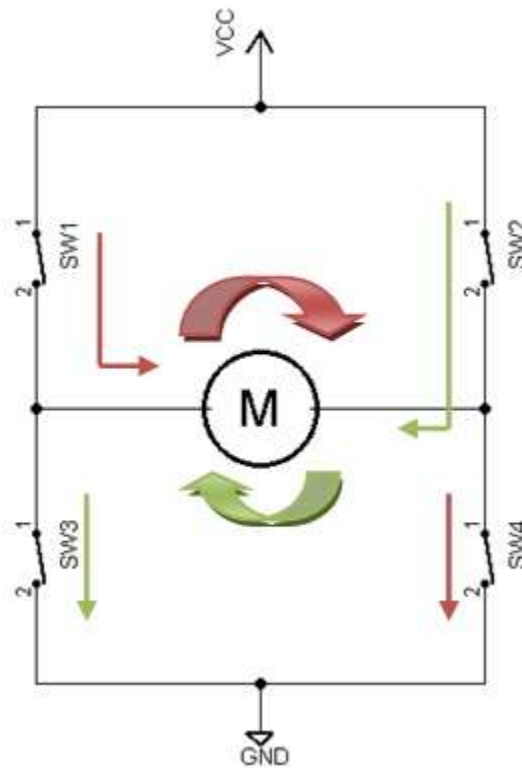


Figure 20. H-Bridge operation.

3.3.2. Introduction to L293D IC

The L293D half-H driver is an integrated motor driver in a 16-pin DIP which it is designed to accept low current signal from microcontroller. It also provides high current and voltage as output in order to operate a motor (Texas Instrument 2002). In other words it acts as an amplifier for motors (Texas Instrument 2014).

To amplify the voltage and current, the L293D includes four Darlington transistors instead of switches which accept a small current and provide current of up to 600mA with 1.2A per channel and voltages from 4.5V to 36V. The L293D has two enable inputs which are associated with two pairs of drivers. When the enable input 1 (EN1,2) is high, the drivers 1 and 2 are active. On the other hand, when the enable input 2 (EN3,4) is high, the drivers 3 and 4 are active. A very important feature of this driver is its internal clamp diodes. (Texas Instrument 2002). The clamp diodes or fly back diodes protect the driver from high voltage transients which are produced by the motor (National Instruments 2013). To protect the IC from overheat, there is a thermal overload protection sensor which stops driving the motor when the temperature increases above the recommended conditions. (Texas Instrument 2002). The inputs and outputs of L293D IC are illustrated in the Figure 21. Table 5 provides more details about the function of pins.

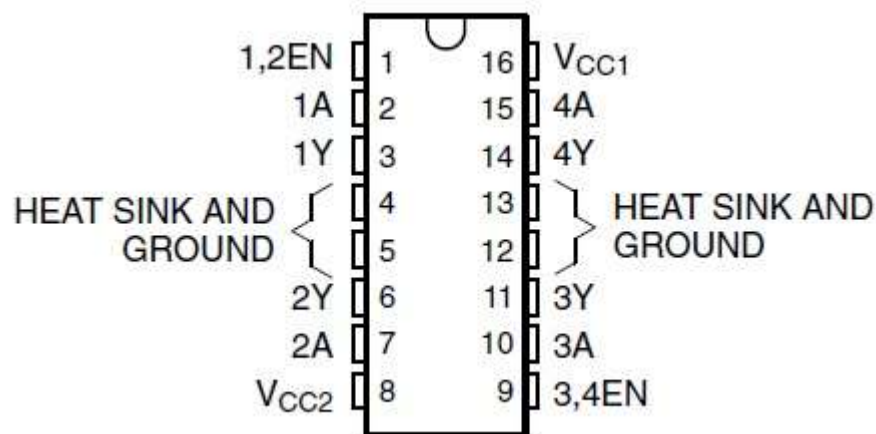


Figure 21. L293D package (Texas Instrument 2002).

Table 5. Pin mapping.

Pin No	Name	Function
1	1,2EN	Enables the driver 1 when it is high.
2	1A	Digital input of driver 1.
3	1Y	Output of driver 1 for
4	GND	motor.
5	GND	Ground.
6	2Y	Ground.
7	2A	Output of driver 1 for
8	VCC2	motor.
9	3,4EN	Digital input of driver 1.
		Supply voltage up to 36V
10	3A	Enables the driver 2 when
11	3Y	it is high.
12	GND	Digital input of driver 2.
13	GND	Output of driver 2 for
14	4Y	motor.
15	4A	Ground
16	VCC1	Ground
		Output of driver 2 for
		motor
		Digital input of driver 2.
		Supply voltage 4,5-7V

3.4. Design and Implementation of L293D board.

To design a board for L293D motor driver, the Cadsoft design software called Eagle is used. The freeware version of Eagle allows use able board area of 100 x 80 mm and only Top and Bottom layers can be used. Those features cover the needs of project. The schematic design starts following the L293D datasheet guidelines. For that purpose the requirement components are:

- PCB Universal board
- L293D motor driver
- 5 row pin header
- two dual row pin header
- cables

The Eagle provides various component libraries making the design process simpler and faster. The Figure 22 illustrates the schematic design of L293D board. On 5-pin header are connected the pins that are related with Arduino Uno board such as, EN input, 1A and 2A inputs, 5Volts VCC1 and the common ground. The M1 and M2 are connected with a DC motor. Finally the last two pins are connected with external power supply at 10Volts. For more details, there is in Appendix 1 the full schematic design of system.

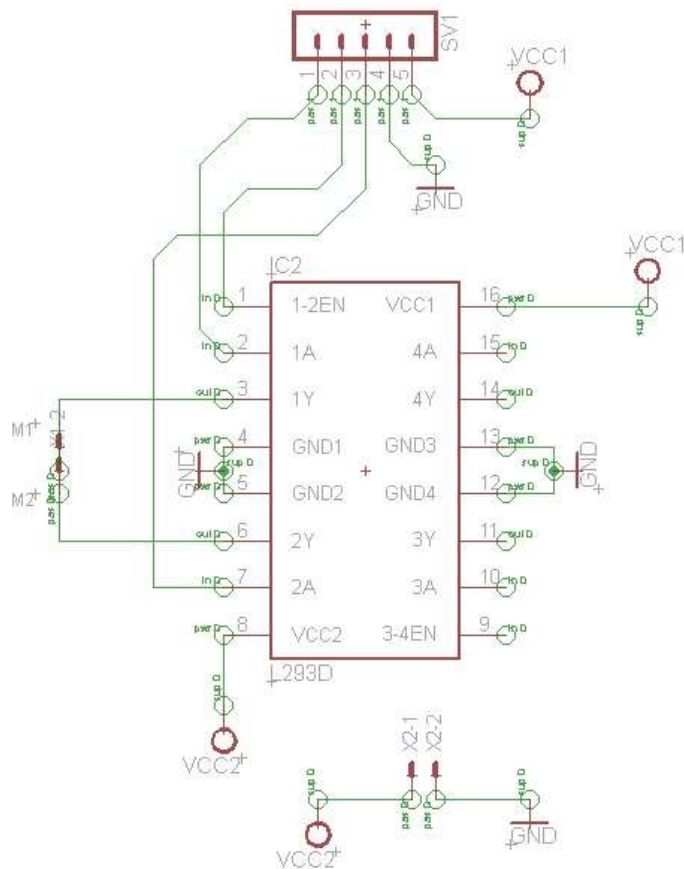


Figure 22. Schematic of L293D board.

Figure 23 presents the implemented L293D PCB board and the connections between components.

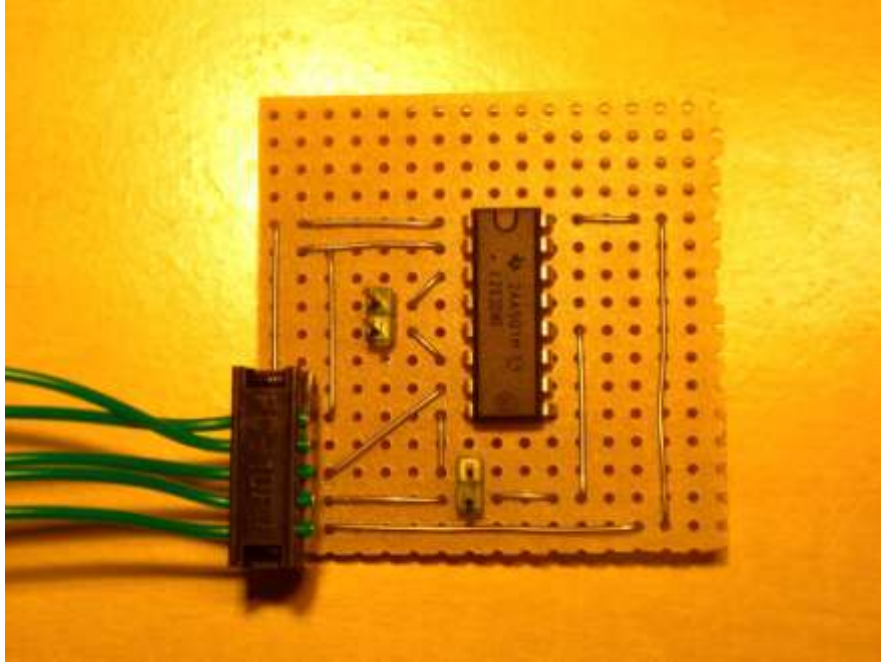


Figure 23. L293D board.

4. SOFTWARE IMPLEMENTATION

4.1. Setting up the Raspberry Pi

Raspberry Pi, according to chapter 2, is a small computer with great capabilities. However, it does not have a BIOS and as a result it boots only from an SD card. To setup the Raspberry Pi board, it requires an SD card of 8 GB size, a laptop, a wireless router, ethernet cables and micro USB power supply. The recommended OS, Raspbian, is available on the official website and is free for downloading. Once the download has finished, the image file should be unzipped and the executable file extracted to the SD card using the Win32DiskImager software (see Figure 24).

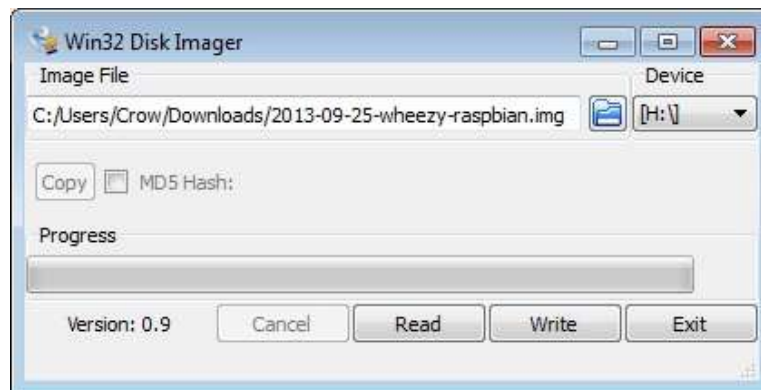


Figure 24. Win32DiskImager.

Once Raspbian has written, the SD card is plugged in back to Raspberry Pi. Due to lack of monitor and keyboard, the access and control of Raspberry Pi are achieved remotely over local network using the Secure Shell (SSH) protocol. The SSH, according to Ylonen and Lonrick (2006) is “a protocol for secure remote login and other secure network services over an insecure network.” One way to connect the Raspberry Pi via SSH is by using an Ethernet cable. Figure 25 illustrates the network.

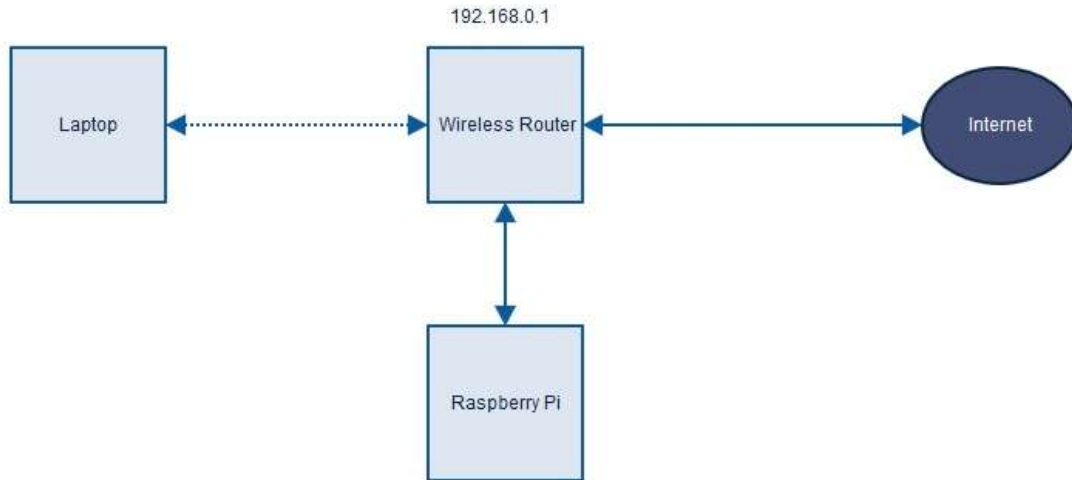


Figure 25. Raspberry remote connection.

Connecting the Raspberry Pi to the network, the router provides a dynamic IP by default. The dynamic IP can change when the Raspberry disconnects from the network. This problem can solve changing the IP from dynamic to static. In order to accomplish that it is necessary to edit the `cmdline.txt` file in SD card and add the IP address as well as the gateway address.

```
ip = 192.168.0.3:::192.168.0.1
```

The SD card has been plugged in back into Raspberry and the board has been powered on. In linux-based system, the user has access to Raspberry via SSH, giving the following command in the terminal.

```
ssh pi@192.168.03
```

```

root@antonis-Lenovo-B590:/home/antonis# ssh pi@192.168.0.3
pi@192.168.0.3's password:
Linux raspberrypi 3.10.25+ #622 PREEMPT Fri Jan 3 18:41:00 GMT 2014 armv6l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri May 30 00:16:35 2014 from 192.168.0.137
pi@raspberrypi ~ $

```

Figure 26. Log in Raspberry Pi.

The Pi specifies the default username and the IP address has already determined in SD card.

The security feature is completed with the default password *raspberry*.

Once the access to Raspberry has been succeed, it is necessary to collect information about the router and Pi by using the following command:

```
ifconfig
```

The next figure shows the information of Ethernet port 0 connection such as the IP address of Raspberry (192.168.0.3), the Broadcast address (192.168.0.255) and the Subnet mask address (255.255.255.0).

```

pi@raspberrypi ~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr b8:27:eb:cd:48:f8
          inet addr:192.168.0.3  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:283 errors:0 dropped:0 overruns:0 frame:0
          TX packets:188 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:23764 (23.2 KiB)  TX bytes:26672 (26.0 KiB)

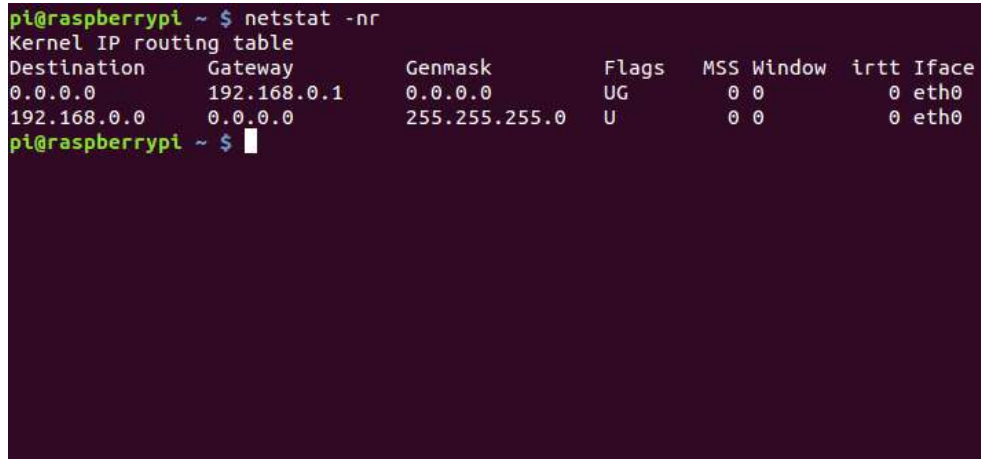
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

```

Figure 27. Ethernet port 0 configurations.

The following command gives more information about the IP routing table.

```
netstat -nr
```



```

pi@raspberrypi ~ $ netstat -nr
Kernel IP routing table
Destination      Gateway         Genmask        Flags   MSS Window  irtt Iface
0.0.0.0          192.168.0.1    0.0.0.0        UG      0  0        0  eth0
192.168.0.0      0.0.0.0        255.255.255.0  U       0  0        0  eth0
pi@raspberrypi ~ $

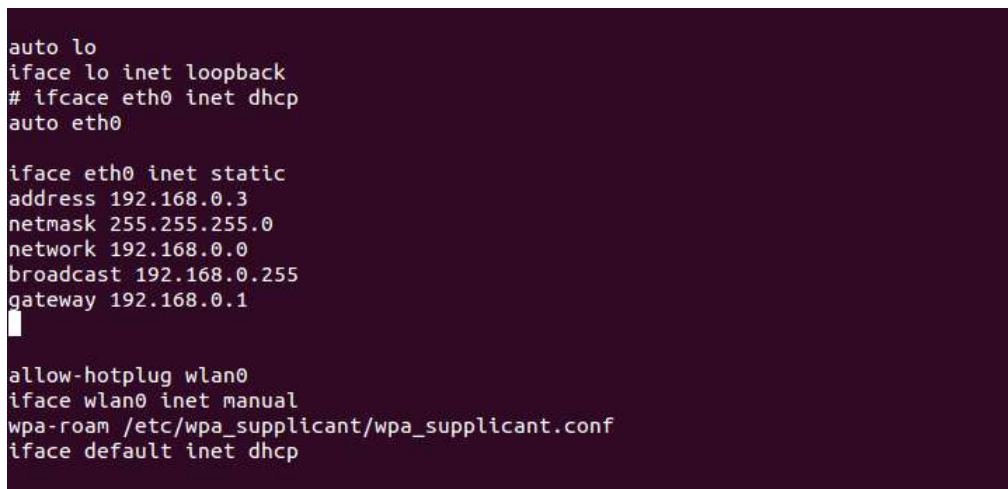
```

Figure 27. IP routing table.

This shows the Gateway address of 192.168.0.1 which connects the outside world with the local network.

The next step is to edit the network configuration with the following command:

```
sudo nano /etc/network/interfaces
```



```

auto lo
iface lo inet loopback
# iface eth0 inet dhcp
auto eth0

iface eth0 inet static
address 192.168.0.3
netmask 255.255.255.0
network 192.168.0.0
broadcast 192.168.0.255
gateway 192.168.0.1

allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp

```

Figure 28. Network interface configuration information.

From the network configuration file the line which enables the DHCP, `iface eth0 inet dhcp`, should be replaced with `iface eth0 inet static`. Below this line, network information should be added as following:

```
address 192.168.0.3

netmask 255.255.255.0

network 192.168.0.0

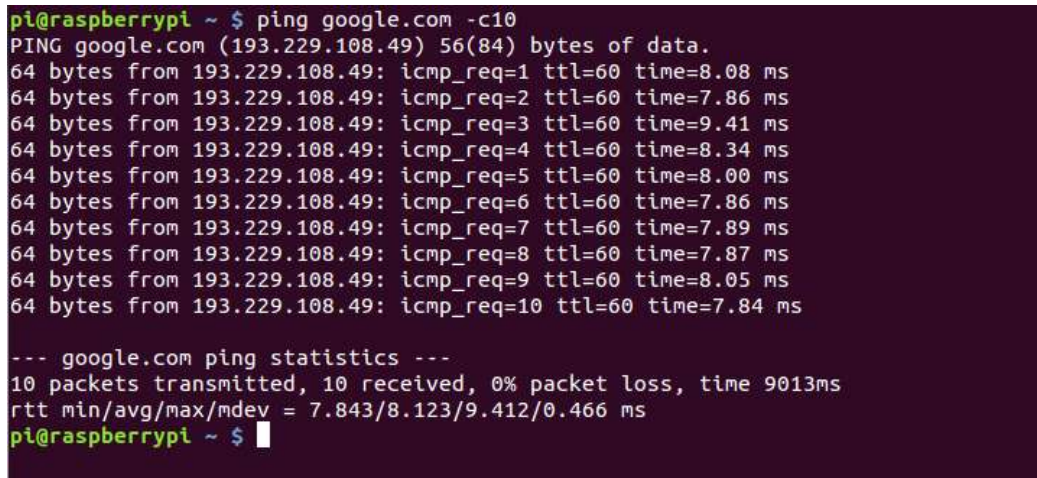
broadcast 192.168.0.255

gateway 192.168.0.1
```

The device should reboot. After logging, the command `ifconfig` is used again to confirm the static ip.

Next step is to check if there is a connection with internet by using the ping command

```
ping google.com -c10
```



```
pi@raspberrypi ~ $ ping google.com -c10
PING google.com (193.229.108.49) 56(84) bytes of data:
64 bytes from 193.229.108.49: icmp_req=1 ttl=60 time=8.08 ms
64 bytes from 193.229.108.49: icmp_req=2 ttl=60 time=7.86 ms
64 bytes from 193.229.108.49: icmp_req=3 ttl=60 time=9.41 ms
64 bytes from 193.229.108.49: icmp_req=4 ttl=60 time=8.34 ms
64 bytes from 193.229.108.49: icmp_req=5 ttl=60 time=8.00 ms
64 bytes from 193.229.108.49: icmp_req=6 ttl=60 time=7.86 ms
64 bytes from 193.229.108.49: icmp_req=7 ttl=60 time=7.89 ms
64 bytes from 193.229.108.49: icmp_req=8 ttl=60 time=7.87 ms
64 bytes from 193.229.108.49: icmp_req=9 ttl=60 time=8.05 ms
64 bytes from 193.229.108.49: icmp_req=10 ttl=60 time=7.84 ms

--- google.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 7.843/8.123/9.412/0.466 ms
pi@raspberrypi ~ $
```

Figure 29. Testing if there is internet connection.

The Figure 29 shows that pinging the google address, all packets has successfully transmitted and received.

Since the Raspberry is connected to internet, the following commands ensure that the operating system is up to date.

```
sudo apt-get update && apt-get upgrade
```

To configure the Raspberry Pi as a web server, it is required to install the Linux Apache MySQL PHP Perl (LAMP) server. It is an open source server suitable for dynamic web pages. This package is suitable for the project's purpose because it contains an Apache web server and PHP. In order to install the LAMP server in Raspberry Pi, it should install the previous components one by one.

To install the Apache web server it needs to type the following command on terminal window:

```
apt-get install apache2
```

Once the installation has completed, PHP5 installs by using the following command:

```
apt-get install PHP5
```

4.2. Software Design of the Raspberry Pi

The following figure shows the flowchart of Python script which runs in the Raspberry Pi. When the Raspberry Pi turns on, the Python script initiates the communication with Arduino Uno board.

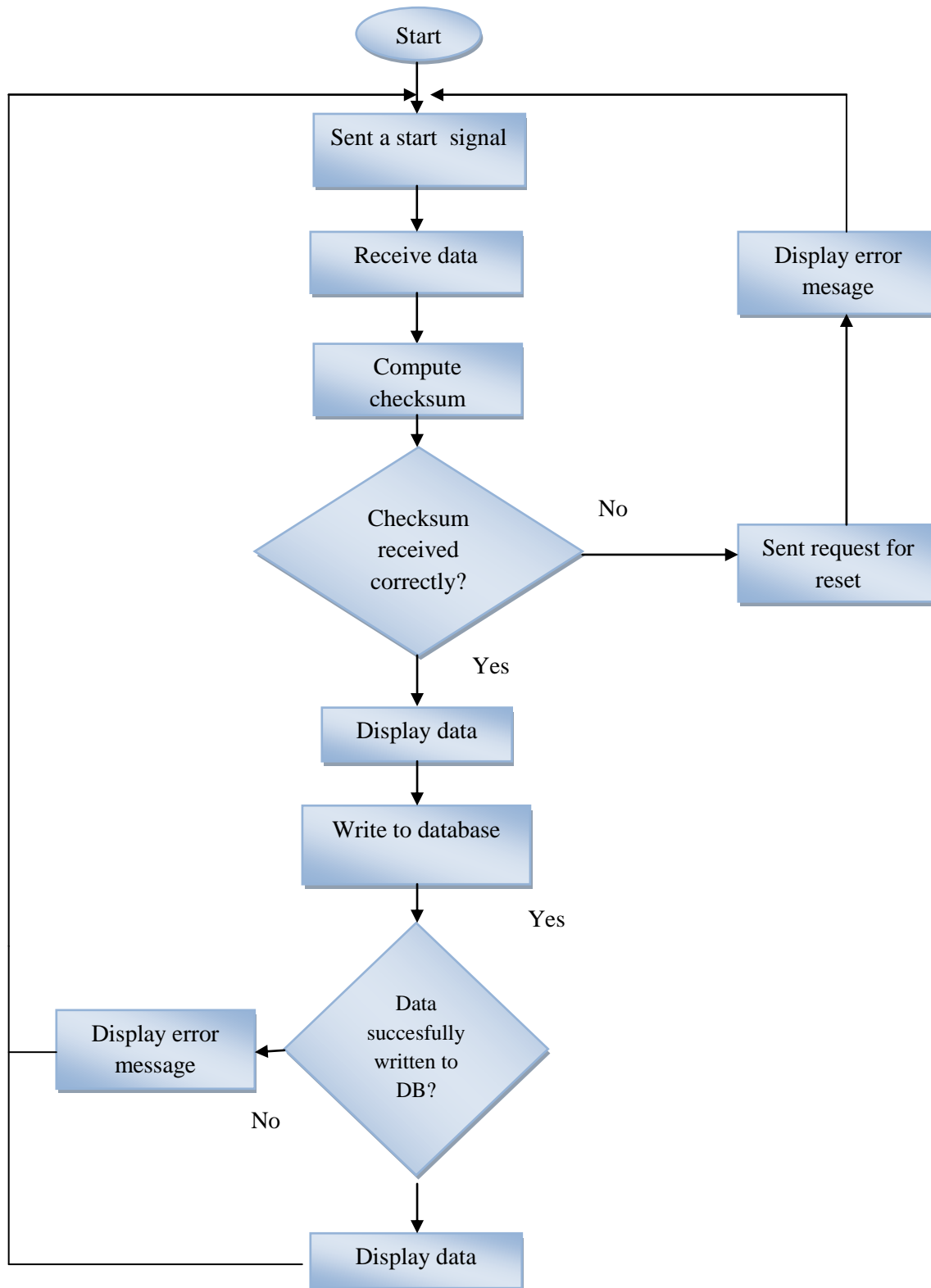


Figure 30. Flowchart of Python script.

When Python script sends a request for data, Arduino Uno sends a response signal of twenty five characters. Without this request from Raspberry Pi, Arduino will not send data. Once it is completed, Raspberry Pi receives 25-bytes of string with sensor measurements. The code computes the checksum of data to detect errors during the transmission. In case, that the data have been received correctly, then measurements such as ID, humidity, temperature, dew point and checksum are displayed in the terminal window. Otherwise, Raspberry sends a request for reset and display error message, thus the procedure initiates again. Once the data have been displayed, the data will be stored in the SQLite database. If the data are not successfully stored in database, an error message will be displayed, otherwise display the data correctly. The whole code are included in the timer loop of ten seconds, thus every ten seconds the database are updated. The full Python script is attached in Appendix 2.

4.3. Software Design of the Arduino Uno

Figure 32 illustrates the flowchart in the Arduino Uno side. The code as already has mentioned is written in Processing programming language. The software opens the serial port and reads the sensor data. When it receives the data request from Raspberry Pi, then Arduino Uno sends a string with the following format:



Figure 31. Data format.

Otherwise it resets the procedure and waits for new request. The sensor DHT11 sends humidity and temperature measurements to Arduino. When the humidity level is lower than 34% then the motor remains idle. If the humidity increases up to 38%, then the motor runs clockwise at 50% of maximum speed, replacing the air inside in the greenhouse with fresh. In case that humidity increases more, the speed increases to 75% of the maximum speed. More than 45% of humidity, the motor stops for 3 seconds and runs again anticlockwise at 100%, removing the air from inside. The previous procedure occurs only when Arduino Uno receives a request from Raspberry Pi.

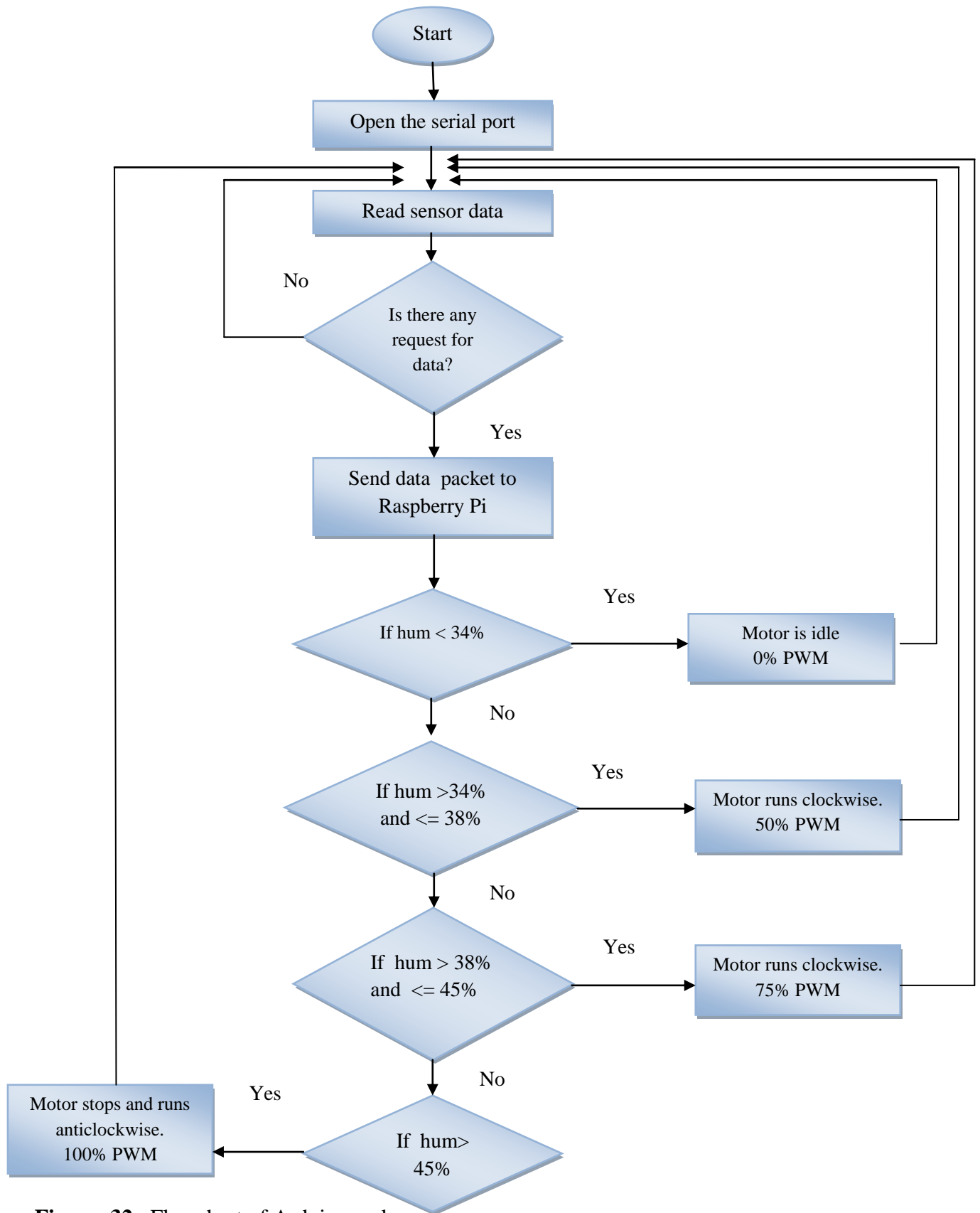


Figure 32. Flowchart of Arduino code.

4.4. Developing a Dynamic Website

4.4.1. Understanding the Website Development

The term of web development is referred to designing and maintaining of a website. A web site can be designed as static or dynamic, depending on the purpose for which it is used. The difference between a static and dynamic website is that the content of static remains always constant. The user send request to server, typing a URL on a browser. The server retrieves the requested page and send it back to user. On the other hand, dynamic web sites have interactive content. In case of dynamic web site, the web server receives a request for data contained in a database. The request is processed by scripts which collect the requested information and send them to user. To build a dynamic website, it is required to use web scripting which is divided in two categories: client-side scripting and server-side scripting. (Rouse 2005, Zandbergen 2014, W3C 2014.)

The client side refers to the user's computer (client) and the displaying content. As the name implies, the server side refers to server's actions. Scripting languages such as Javascript, Action Script, Dart are involved to the client side. Those scripts are often embedded within html code or in separate files. In contrast, the server side contains scripts written in PHP, Perl, Python, Ruby, ASP.NET etc. The scripts are executed in server side before the server response to the user. During the execution the content of scripts is hidden and only the generated file is visible to the user. (Zandbergen 2014, W3C 2014.)

Figure 33 indicates the process of a dynamic web page. More precisely, the user via browser, requests a dynamic page over HTTP. The web server receives the HTTP request and passes it to application server. The request is processed and interpreted by the script (PHP). The PHP script creates a SQL database connection and retrieves the reference data. The application server generates a file with the requested data and passed it to web server. Finally, the file is sent back to user's browser by the server.

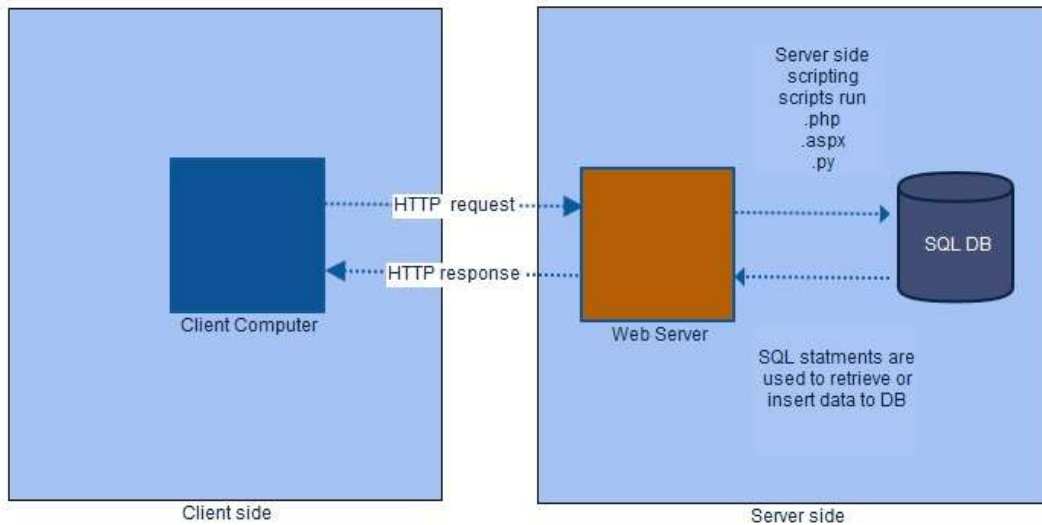


Figure 33. The process of a dynamic web page.

4.4.2. Website Design

The website design is needed to access the data from almost every place in the world. The user is able to monitor the greenhouse condition in real time, through the dynamic website. To develop a website, a good design is required. The design of this website is illustrated in Figure 34. The next step is to create and organize the folders and files into LAMPP server as following:

- Image files
- Css files
 - style.css
- index.php
- header.php
- table.php

- about.php
- contact.php
- footer.php

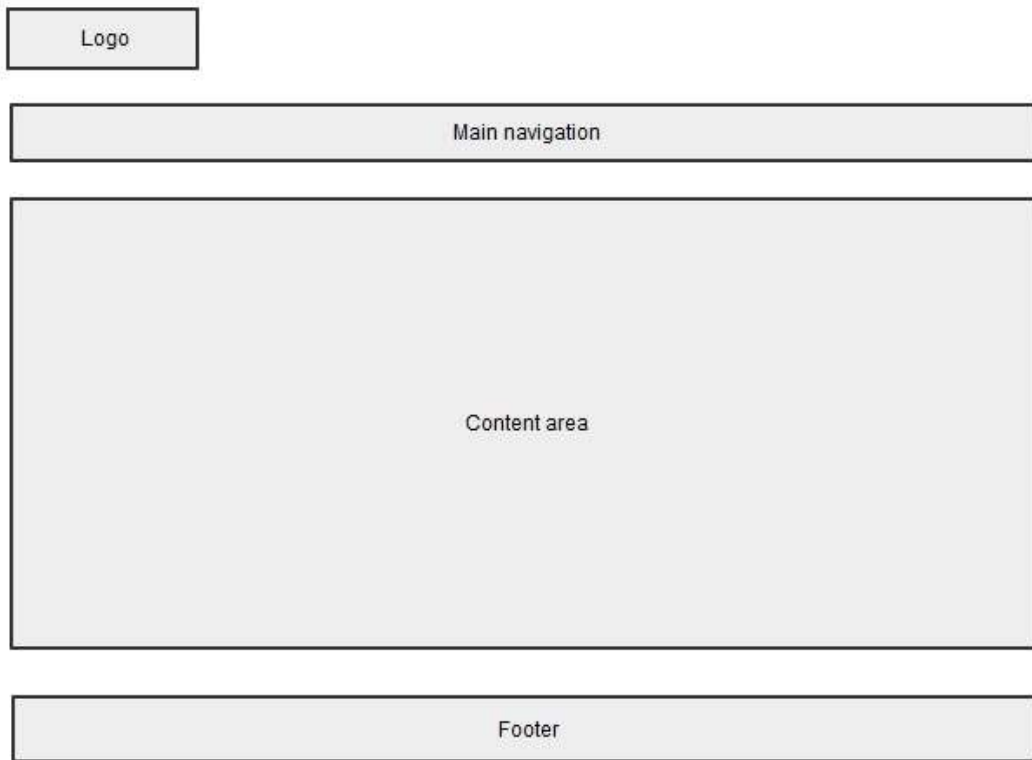


Figure 34. The project plan.

Each of these files are connected with each other using references. In general, the website is designed by using HTML and CSS languages in the client side and PHP with SQL in the server side. HTML and CSS define the appearance and the layout of the website. The PHP scripts define which parts are static and which are dynamic. In this case, only the content can be changed every time where the new page loads. In addition, PHP scripts to create the connection with the SQLite database and to display the data, accessed from the database.

Starting from the top to the bottom, the first block is the header. This area is composed of the navigation menu and the logo.

Regarding the navigation menu, it is divided in the list of four groups which are linked with different PHP pages.

- Home (index.php)
- TableData (table.php)
- About (about.php)
- Contact (contact.php)

The most essential group of those is the Table Data. The goal of this page is to retrieve the data from the SQLite database and to display them on a scrolling table. Figure 35 presents the flowchart of PHP SQLite code. Once the table is created with HTML code, a script in PHP is responsible to initiate the communication with the SQLite database. The script opens the database according to the path of .db file. In case that the database does not exist, an error message is displayed. Otherwise data from the sensor DHT11 table is retrieved and displayed on the website. The table is updated as long the database is open.

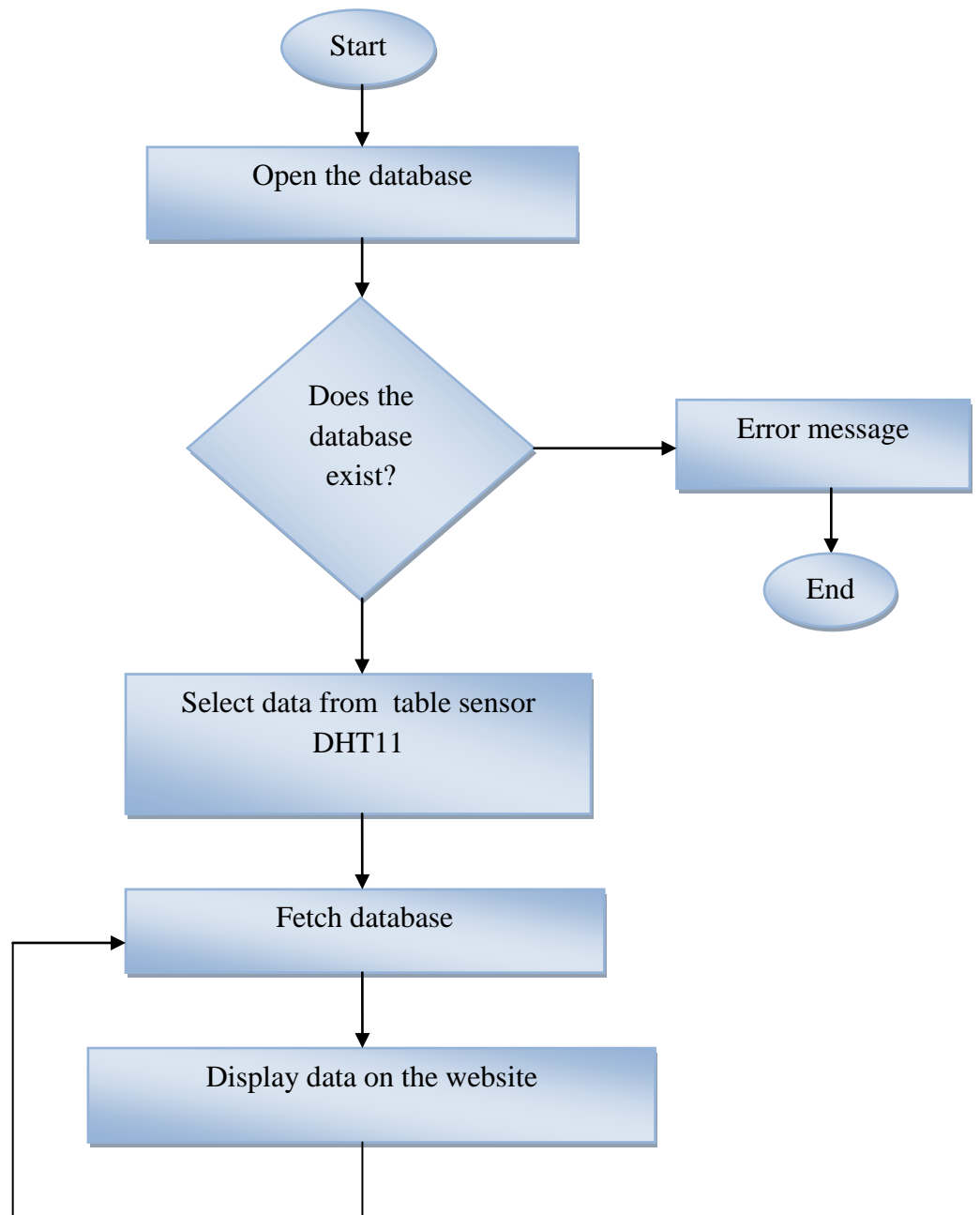


Figure 35. PHP-SQLite3 flowchart.

5. SYSTEM TESTING

Once the hardware and software have been implemented, the system is needed to evaluate and resolve any potential errors. Figure 36 presents the wiring connections between modules.

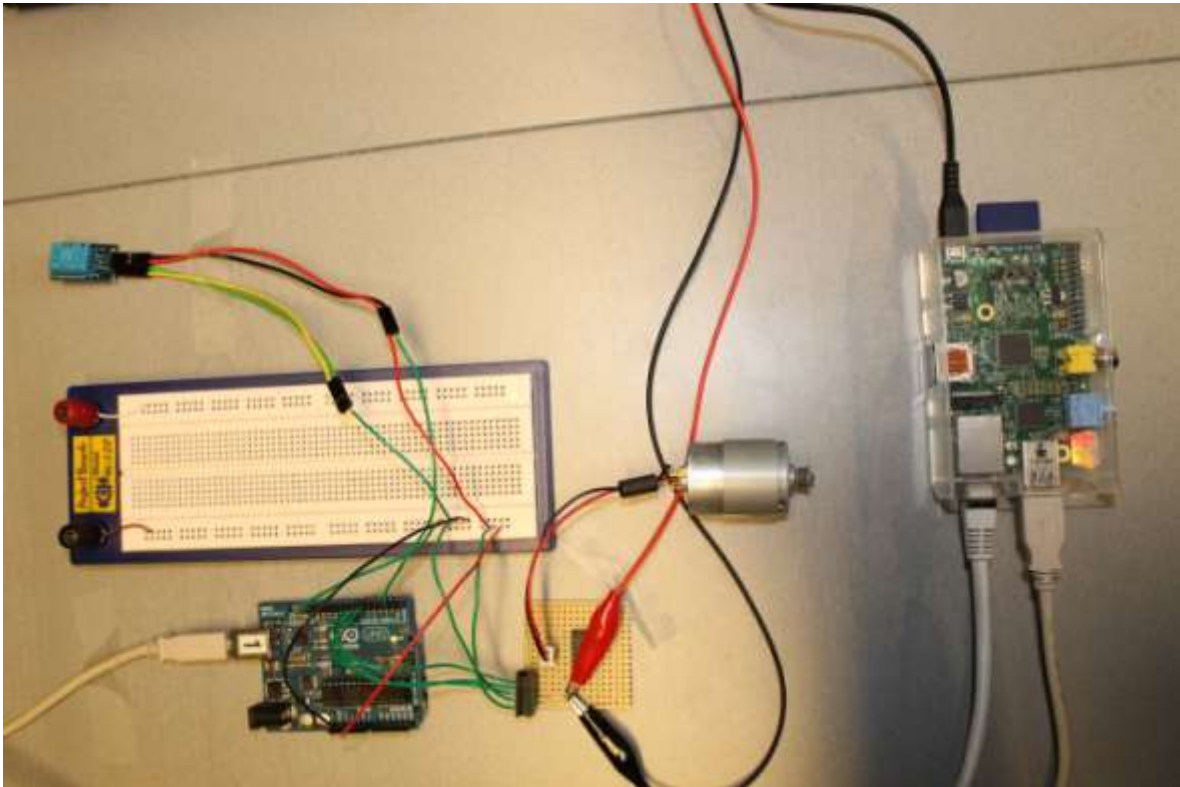


Figure 36. Wiring connections.

The system begins the procedure of data collection and display the data on the website (see Figure 40). To observe the PWM waveform, it requires a digital oscilloscope which is connected with pin 6 of the Arduino board and the common ground. If the humidity levels increase over 34%, the motor runs clockwise at 50% of maximum speed. Figure 37 illustrates the waveform of 50% PWM.

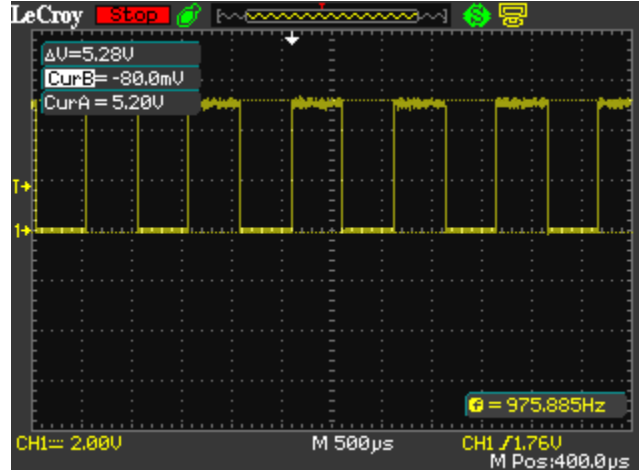


Figure 37. 50% PWM.

When the humidity levels reach up to 45%, the motor runs faster at 75% of the maximum speed. Figure 38 indicates the waveform of 75% PWM.

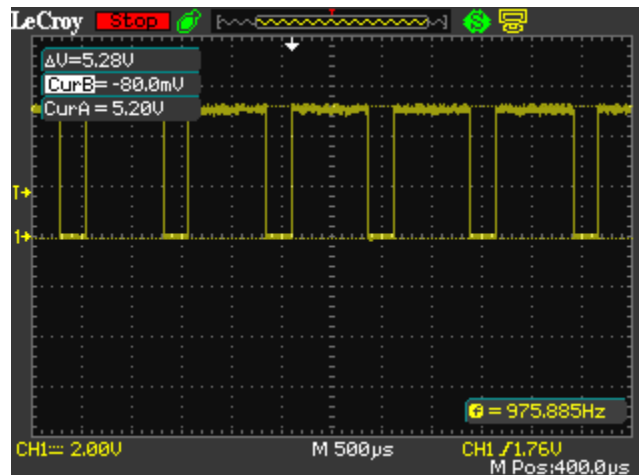


Figure 38. PWM 75%.

Over 45% of relative humidity, the DC motor runs anticlockwise at maximum speed. Figure 39 presents the waveform of 100% PWM.

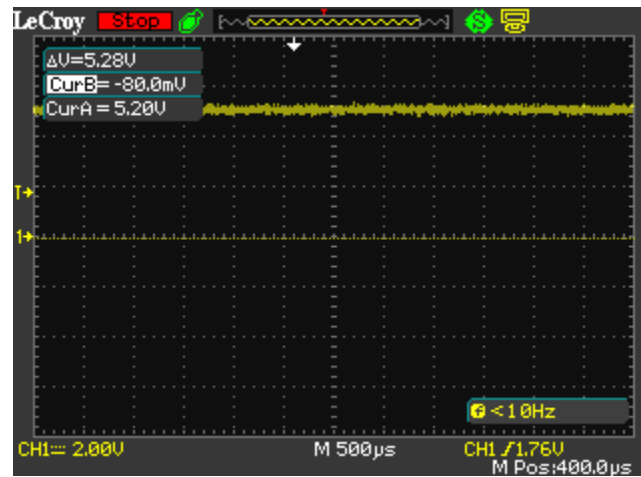


Figure 39. PWM 100%.

In order to use the graphic environment of Raspberry Pi, the use of Xming server and Putty SSH client on Windows 7 was required. Working on the graphical user interface (GUI), the web browser can be used for the connection with the web site.

The Figure 40 illustrates the designed and implemented website that displays the data (humidity, temperature and dew point) from the SQLite database of the Raspberry PI.

SCHEMA

HOME TABLEDATA ABOUT CONTACT

Sensor Data Table

ID	TEMPERATURE	HUMIDITY	DEWPOINT	TIMESTAMP
1	34	26	8	2014-06-06 11:57:04
1	34	26	8	2014-06-06 11:57:23
1	34	26	8	2014-06-06 11:58:10
1	40	31	13	2014-06-06 11:58:40
1	74	35	29	2014-06-06 11:59:13
1	35	27	10	2014-06-06 12:00:24
1	34	27	9	2014-06-06 12:00:54
1	34	27	9	2014-06-06 12:01:24
1	63	35	26	2014-06-06 12:01:57
1	70	32	23	2014-06-06 12:02:27
1	69	30	23	2014-06-06 12:03:07
1	68	28	21	2014-06-06 12:03:27
1	67	28	21	2014-06-06 12:03:57
1	66	27	20	2014-06-06 12:10:27
1	63	27	19	2014-06-06 12:10:58

HOME TABLEDATA ABOUT CONTACT

©2014 Schema

Figure 40. The table of data.

6. EXPERIMENTS

In this section the power consumption of system has been measured and analyzed. Table 6 presents the measurements that have been taken, using a digital multimeter. Starting from the DHT11 sensor, in idle state the current consumption has average value at $111\mu\text{A}$ and maximum value at $116\mu\text{A}$. On the other hand, the DC motor requires more current, because of variety of speeds. More precisely, the motor is power by external power source at 7 Volts, consumes 7.6mA when is stopped. The consumption increased as the speed is increased. Thus, the motor consumes 62.6mA at 50% of maximum speed, 74.8mA at 75% and 83.5mA at maximum speed.

Table 6. Power consumption measurements.

Component	Current (I)	Voltage (V)	Power (P)	Mode
DHT11 sensor	$116\mu\text{A}$	5V	0.00058W	Active
Motor	7.6mA	7V	0.0532W	Idle
	62.6mA	7V	0.438W	50% PWM
	74.8mA	7V	0.523W	75% PWM
	83.5mA	7V	0.584W	100% PWM
Arduino UNO	68.4mA	7V	0.478W	Active
Raspberry Pi	433mA	5V	2.165W	Idle
	436mA	5V	2.18W	Active

To measure the consumption of Arduino board, an external power supply is used at 7 Volts. Arduino Uno consumes 68,4mA providing to peripherals enough power. Raspberry Pi, Model B can use 1A maximum (Raspberry Pi Foundation 2014d). According to

measurements, the Raspberry Pi consumes 433mA when is idle. This value increases at 436mA when the implemented software for temperature and humidity monitoring is running.

Once the current has been measured, the power can be calculated according to the following formula:

$$P(t) = I(t) \times V(t) \quad (2)$$

P is the power measured in watts, I is the current measured in amperes and V is the voltage measured in volts. (Wikipedia 2014c.)

It can be noticed that the power consumption of the system in this experiment is low, almost 3 Watts. Having the system low power consumption, efficiency and stability increase, reducing at the same time the system's temperature and the cost of maintenance.

7. CONCLUSION AND FUTURE WORK

The goal of this thesis was to design and implement an inexpensive and low-powered monitoring system for greenhouses. In order to achieve that, two of the most popular electronic boards and a number of programming languages have been used. Using the SQLite database, the system is capable to store the information received from the sensor. A web server application provides a link between user and database, allowing the greenhouse monitoring through a dynamic web site. A DC motor acting as ventilator is used to change the air flow direction according to humidity levels. To design this system, the knowledge of software and hardware implementation was required. For the database implementation, a good skills of Python, Arduino language, SQL and PHP were needed

The system can be implemented in small size greenhouses with specific modifications in the hardware part, according on the needs of the greenhouse. In order to monitor a large structure, intermediate electronic parts have to alter to achieve high power supply to the ventilator.

In the future, the system may be improved with wireless modules and electronic components capable to provide high current for ventilation system operation. In addition, the web site can be improved by using responsive web design, giving to user the opportunity to monitor the greenhouse, through a wide range of access methods.

REFERENCES

- Arch Linux ARM (2014). *Arch Linux* [Online]. archlinuxarm.org [cited 21 Mar. 2014]. Available from the World Wide Web: <URL: <http://www.archlinuxarm.org>>.
- ARM Ltd. *ARM11 Processor Family* [online].arm.com [cited 20 Mar. 2014]. Available from the World Wide Web: <URL:<http://www.arm.com/products/processors/classic/arm11/index.php>>.
- Arduino (2014a). *Introduction* [Online]. arduino.cc [cited 13 April. 2014]. Available from the World Wide Web: < URL:<http://www.arduino.cc/en/Guide/Introduction>>.
- Arduino (2014b). *Arduino Uno* [Online]. Arduino.cc [cited 12 April 2014]. Available from the World Wide Web: <URL: <http://arduino.cc/en/Main/arduinoBoardUno>>.
- Arduino (2014c). *What Arduino can do* [Online].arduino.cc [cited 13 April. 2014]. Available from the World Wide Web: < URL:<http://arduino.cc/>>.
- Arduino (2014d). *Libraries* [Online]. arduino.cc [cited 14 April. 2014]. Available from the World Wide Web :< URL:<http://arduino.cc/en/Reference/Libraries>>.
- Atmel Corporation (2014a). *ATmega328P Datasheet* [Online]. Atmel.com [cited 14 April 2014]. Available from the World Wide Web: <URL:http://www.atmel.com/Images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet.pdf>.
- Atmel Corporation (2014b). *picoPower Technology* [Online]. Atmel.com [cited 14 April 2014]. Available from the World Wide Web: <URL:<http://www.atmel.com/Technologies/lowpower/picopower.aspx>>.

- Banzi, Massimo (2011). *Getting Started with Arduino*. 2nd Ed. Sebastopol: O'Reilly
- Bowling, Sue Ann (1987). *How Do Greenhouses Work?* [Online]. USA: University of Alaska Fairbanks [cited 23 Feb. 2014]. Available from the World Wide Web: <URL:<http://www2.gi.alaska.edu/ScienceForum/ASF8/817.html>>.
- Both, A.J (2008). *Greenhouse Temperature Management* [online].New Jersey, USA: Rutgers University [cited 23 Feb. 2014]. Available from the World Wide Web:
<URL:<http://njveg.rutgers.edu/assets/pdfs/ajb/Temperature%20Management.pdf>>.
- Bernstein, Philip and Eric Newcomer (2009). *Principles of Transaction Processing*. 2ndEd. Burlington, USA: Morgan Kaufmann Publishers.
- Badgery-Parker, Jeremy (1999). *The greenhouse* [online].1st Ed. Gosford, Australia: NSW Department of Primary Industries [cited 13 Feb. 2014]. Available from the World Wide Web:
<URL:http://www.dpi.nsw.gov.au/__data/assets/pdf_file/0008/119348/greenhouses.pdf>.
- Brash, David (2002). *The ARM Architecture Version 6 (ARMv6)* [online].ARM Ltd [cited 20 Mar. 2014]. Available from the World Wide Web:
<URL:http://itu.dk/courses/ISOM/E2005/ARMv6_Architecture.pdf>.
- Broadcom (2014). *BCM2835 High Definition 1080p Embedded Multimedia Applications Processor* [online].broadcom.com [cited 29 Mar. 2014]. Available from the World Wide Web:
< URL:<http://www.broadcom.com/products/BCM2835>>.
- Bakker, J.C. (2001). *Greenhouse Climate Control: An Integrated Approach*. Netherlands: Wageningen Academic Publishers.

- Cormie, David (2002). *The ARM11™ Microarchitecture* [online]. ARM Ltd [cited 20 Mar 2014]. Available from the World Wide Web:
<URL:http://www.arm.com/support/White_Papers>.
- Castilla, Nicolas (2013). *Greenhouse Technology and Management*. 2nd Ed. Wallingford, UK: CABI. ISBN: 978178064 1034.
- CENELEC (2011). *New standard for common mobile chargers*. Brussels: European Committee for Electrotechnical Standardization.
- Dennis, Andrew (2013). *Raspberry Pi Home Automation with Arduino*. UK: Packt Publishing Ltd. ISBN 978-1-84969-586-2.
- Elinux (2014a). *RPi Low-level peripherals* [online]. elinux.org [cited 27 Mar. 2014]. Available from the World Wide Web:
<URL: http://elinux.org/RPi_Low-level_peripherals>.
- Elinux (2014b). *RPi Hardware* [online]. elinux.org [cited 27 Mar. 2014]. Available from the World Wide Web :< URL:http://elinux.org/RPi_Hardware>.
- Eshenaur, Brian and Robert Anderson (2004). *Managing the Greenhouse Environment to Control Plant Diseases* [online]. USA: University of Kentucky [cited 23 Feb. 2014]. Available from the World Wide Web:
<URL:http://www2.ca.uky.edu/agcollege/plantpathology/ext_files/PPFShtml/PPFS-GH-1.pdf>.
- Grant, Allen and Mike Owens (2010). *The Definitive Guide to SQLite*. 2nd Ed. New York: Apress. ISBN-13 978-1-4302-3225-4.
- Halfacree, Gareth (2012). *Raspberry Pi Interview: Eben Upton reveals all* [online]. linuxuser.co.uk [cited 17 Mar.2014]. Available from the World Wide Web:
<URL:<http://www.linuxuser.co.uk/features/raspberry-pi-interview-eban-upton-reveals-all>>.
- Horan, Brendan (2013). *Practical Raspberry Pi*. USA: Apress Media.

- HDMI (2014a). *What is the difference between DVI and HDMI?* [Online].hdmi.org [cited 20 Mar. 2014].Available from the World Wide Web:
<URL:<http://www.hdmi.org/learningcenter/kb.aspx#83>>.
- HDMI (2014b). *What is CEC?* [Online].hdmi.org [cited 20 Mar. 2014]. Available from the World Wide Web:
<URL:<http://www.hdmi.org/learningcenter/kb.aspx#83>>.
- Kreibich, Jay (2010). *Using SQLite*.1st Ed. Sebastopol, USA: O’ Reilly. ISBN: 978-0-596-52118-9.
- Linke, Bernhard (2008). *Overview of 1-Wire Technology and Its Use* [online]. Maxim Integrated [cited 10 Mar. 2014]. Available from the World Wide Web :
<URL:<http://www.maximintegrated.com/app-notes/index.mvp/id/1796>>.
- Linke, Bernhard (2009). *Reading and Writing 1-Wire Devices Through Serial Interfaces* [online]. Maxim Integrated [cited 10 Mar. 2014]. Available from the World Wide Web:
<URL:<http://www.maximintegrated.com/app-notes/index.mvp/id/74>>.
- Lipovski, Jack (1999). *Single-and Multi-Chip Microcontroller Interfacing For the Motorola 68HC12*. California, USA: Academic Press.
- Littmark, Fanny (2013). *How the Greenhouses Effect Works* [Online]. Comsol. Inc.[cited 24 February 2014]. Available from the World Wide Web:
<URL: <http://www.comsol.com/blogs/the-greenhouse-effect/>>.
- Microchip Technology (2012). *USB 2.0 Hub and 10/100 Ethernet Controller* [online].microchip.com [cited 12 Mar.2014]. Available from the World Wide Web:
<URL:<http://ww1.microchip.com/downloads/en/DeviceDoc/9512.pdf>>.

- Maxim Integrated (2014). *I-Wire Tutorial Presentation* [online]. maximintegrated.com [cited 12 Mar. 2014]. Available from the World Wide Web:
<URL:<http://www.maximintegrated.com/products/1-ire/flash/overview/index.cfm>>.
- McColl, Nicolas (2002). *Temperature and Dew Point Tutorial* [Online].USA: Lyndon State College, Department of Atmospheric Sciences. Available from the World Wide Web:
<URL:http://apollo.lsc.vsc.edu/classes/idm3020/tut_folder/nick_tutorial/>.
- McManis, Chuck (2006). *H-Bridges: Theory and Practice* [Online]. mcmanis.com [cited 15 May. 2014]. Available from the World Wide Web:
<URL: <http://www.mcmanis.com/chuck/robotics/tutorial/h-bridge/>>.
- National Instruments (2013). *Protecting NI Switch Modules when Switching Inductive Loads* [Online].ni.com[cited 15 May 2014]. Available from World Wide Web:
<URL:<http://digital.ni.com/public.nsf/allkb/AB895F6C52DD73D3862573F3007FBB7C>>.
- Nave, C.R. (2012a). *Greenhouse Effect* [online].USA: Georgia State University [cited 23 Feb. 2014]. Available from the World Wide Web:
< URL:<http://hyperphysics.phy-astr.gsu.edu/hbase/thermo/grnhse.html>>.
- Nave, C.R. (2012b). *Heat Convection* [online].USA: Georgia State University [cited 23 Feb. 2014]. Available from the World Wide Web:
<URL:<http://hyperphysics.phy-astr.gsu.edu/hbase/thermo/heatra.html#c2>>.
- Nave, C.R. (2012c). *Relative Humidity* [online].USA: Georgia State University [cited 23 Feb. 2014]. Available from the World Wide Web:
<URL:<http://hyperphysics.phy-astr.gsu.edu/hbase/kinetic/relhum.html#c1>>.
- Nave, C.R. (2012d). *Dew point* [online].USA: Georgia State University [cited 23 Feb. 2014]. Available from the World Wide Web:
<URL:<http://hyperphysics.phy-astr.gsu.edu/hbase/kinetic/relhum.html#c2>>.

- NSW Government. *Ventilation in Greenhouses* [online]. Australia: NSW Department of Primary Industries [cited 23 Feb. 2014]. Available from the Worldwide Web: <URL:<http://www.dpi.nsw.gov.au/agriculture>>.
- Raspberry Pi Foundation (2014a). *The Making of Pi* [online].raspberrypi.org [cited 16 Mar.2014].Available from the World Wide Web:
<URL:<http://www.raspberrypi.org/about/>>.
- Raspberry Pi Foundation (2014b). *What is the difference between Model A and Model B?* [Online]. raspberrypi.org [cited 16 Mar. 2014]. Available from Web Wide Web:< URL: <http://www.raspberrypi.org/help/faqs/#generalDifference>>.
- Raspberry Pi Foundation (2014c). *What operating system (OS) does it use?* [Online]. raspberrypi.org [cited 16 Mar. 2014]. Available from Web Wide Web: < URL: <http://www.raspberrypi.org/help/faqs/#softwareOS>>.
- Raspberry Pi Foundation (2014d). *What are the power requirements?* [Online]. raspberrypi.org [cited 11 Jun. 2014]. Available from Web Wide Web: < URL: <http://www.raspberrypi.org/help/faqs/#performanceSpeed>>.
- Raspbian (2014).*What is Raspbian?* [Online].raspbian.org [cited 16 Mar. 2014]. Available from Web Wide Web:
<URL: http://www.raspbian.org/RaspbianFAQ#What_is_Raspbian.3F>.
- Red Hat (2014). *Pidora* [Online]. Fedoraproject.org [cited 3 May 2014]. Available from Web Wide Web: <URL: <http://www.fedoraproject.org/wiki/Remix>>.
- Richardson, Matt and Shawn Wallace (2013). *Getting Started with Raspberry Pi*. USA: O'Reilly Media. ISBN: 978-1-449-34421-4.
- Sibsankar, Haldar (2007). *Inside SQLite*. 1st Ed. Sebastopol, USA: O' Reilly. ISBN-13: 978-0-59-655006-6.

Rouse, Margaret (2005). *Dynamic and static* [Online]. Searchingnetworking.techtarget.com [cited 4 May 2014]. Available from Web Wide Web: <URL:<http://searchnetworking.techtarget.com/definition/dynamic-and-static>>.

SQLite (2014a). *About SQLite* [online]. Sqlite.org [cited 25 Feb. 2014]. Available from the World Wide Web: <URL: [https:// www.sqlite.org/about.html](https://www.sqlite.org/about.html) >.

SQLite (2014b). *Size of the SQLite Library* [online]. Sqlite.org [cited 25 Feb. 2014]. Available from the World Wide Web: <URL: [https:// www.sqlite.org/footprint.html](https://www.sqlite.org/footprint.html) >.

SQLite (2014c). *SQLite Copyright* [online]. Sqlite.org [cited 25 Feb. 2014]. Available from the World Wide Web: <URL: [https:// www.sqlite.org/copyright.html](https://www.sqlite.org/copyright.html) >.

SQLite (2014d). *Transactional* [online]. Sqlite.org [cited 25 Feb. 2014]. Available from the World Wide Web: <URL: [https:// www.sqlite.org/transactional.html](https://www.sqlite.org/transactional.html) >.

SQLite (2014e). *Locking* [online]. Sqlite.org [cited 25 Feb. 2014]. Available from the World Wide Web:< URL: [https:// www.sqlite.org/locking3.html](https://www.sqlite.org/locking3.html) >.

SQLite (2014f). *Atomic Commit* [online]. Sqlite.org [cited 25 Feb. 2014]. Available from the World Wide Web: <URL: [https:// www.sqlite.org/atomiccommit.html](https://www.sqlite.org/atomiccommit.html) >.

Sammons, Philip J., Tomonari Furukawa and Andrew Bulgin (2005). *Autonomous Pesticide Spraying Robot for use in a Greenhouse* [online]. Australia: University of New South Wales. Available from the World Wide Web: <URL:<http://www.cse.unsw.edu.au/~acra2005/proceedings/papers/sammons.pdf>>

Shibu, Tekijät (2009). *Intro to Embedded Systems*. 1st Ed. New Delhi: Tata McGraw-Hill.

- Silicon Labs (2014). *Serial Communications* [online].silabs.com [cited 12 Mar.2014]. Available from the World Wide Web:<URL: http://www.silabs.com/Support%20Documents/Software/Serial_Communications.pdf>.
- Summerville, Douglas (2009). *Embedded Systems Interfacing for Engineers using the Freescale HCS08 Microcontroller I: Assembly Language Programming*.1st Ed. Morgan and Claypool Publishers.
- SparkFun Electronics (2012). *Serial Communication* [online]. sparkfun.com [cited 13 Mar.2014]. Available from the World Wide Web: <URL:<https://learn.sparkfun.com/tutorials/serial-communication>>.
- Sjogelid, Stefan (2013). *Raspberry Pi for Secret Agents*.UK: Packt Publishing. ISBN 978-1-84969-578-7.
- Texas Instrument (2002). *L293, L293D Quadruple Half-H Drivers Datasheet* [Online].ti.com [cited 15May 2014]. Available from World Wide Web: <URL: www.ti.com/lit/ds/symlink/l293d.pdf>.
- Texas Instrument (2014a). Motor Drive and Control [Online].ti.com [cited 15May 2014]. Available from World Wide Web: <URL:http://www.ti.com/llds/ti/apps/motor/products/integrated_motor_drivers.page>.
- Upton, Eben (2011). *Power Supply confirmed as 5V MicroUSB* [online]. raspberrypi.org [cited 25 Mar. 2014]. Available from the World Wide Web: <URL: <http://www.raspberrypi.org/power-supply-confirmed-as-5v-micro-usb/>>.
- Wallen, Jack (2010). *Easy LAMP Server Installation* [Online]. Linux.com [cited 30 May 2014]. Available from the World Wide Web: <URL:<http://www.linux.com/learn/tutorials/288158-easy-lamp-server-installation>>.

- Willey, Michael (2001). *One Cheap network Topology* [online]. Embedded.com [cited 14 Mar.2014]. Available from the World Wide Web:
<URL:<http://www.embedded.com/design/connectivity/4023295/One-Cheap-Network-Topology>>.
- Wheat, Dale (2011). *Arduino Internals*. 1st Ed. New York, USA: Apress.
- Wikipedia (2014a). *Precipitation* [Online].wikipedia.org [cited 15 Feb. 2014]. Available from the World Wide Web:
< URL:<http://en.wikipedia.org/wiki/Precipitation>>.
- Wikipedia (2014b). *Water cycle* [Online].wikipedia.org [cited 15 Feb. 2014]. Available from the World Wide Web:
< URL:http://en.wikipedia.org/wiki/Water_cycle>.
- Wikipedia (2014c). *Power* [Online].wikipedia.org [cited 11 June. 2014]. Available from the World Wide Web:
< URL: http://en.wikipedia.org/wiki/Power_%28physics%29>.
- W3C (2014). *How does the Internet work* [Online]. w3c.com [cited 4 June 2014]. Available from the World Wide Web:
<URL: http://www.w3.org/wiki/How_does_the_Internet_work>.
- Ylonen, T. and Ed. Lonvick (2006). *The Secure Shell (SSH) Authentication Protocol*. [Online]. The Internet Engineering Task Force (IETF) [cited 29 May 2014]. Available from the World Wide Web:
<URL:<http://tools.ietf.org/html/rfc4252>>.
- Zandbergen, Paul (2014). *Web Scripting: Client-side and Server-side* [Online]. Educational portal.com [cited 4 June 2014]. Available from the World Wide Web:
<URL: <http://education-portal.com/academy/lesson/web-scripting-client-side-and-server-side.html#lesson>>.

APPENDIXES

APPENDIX 1.

The overall schematic of the system.

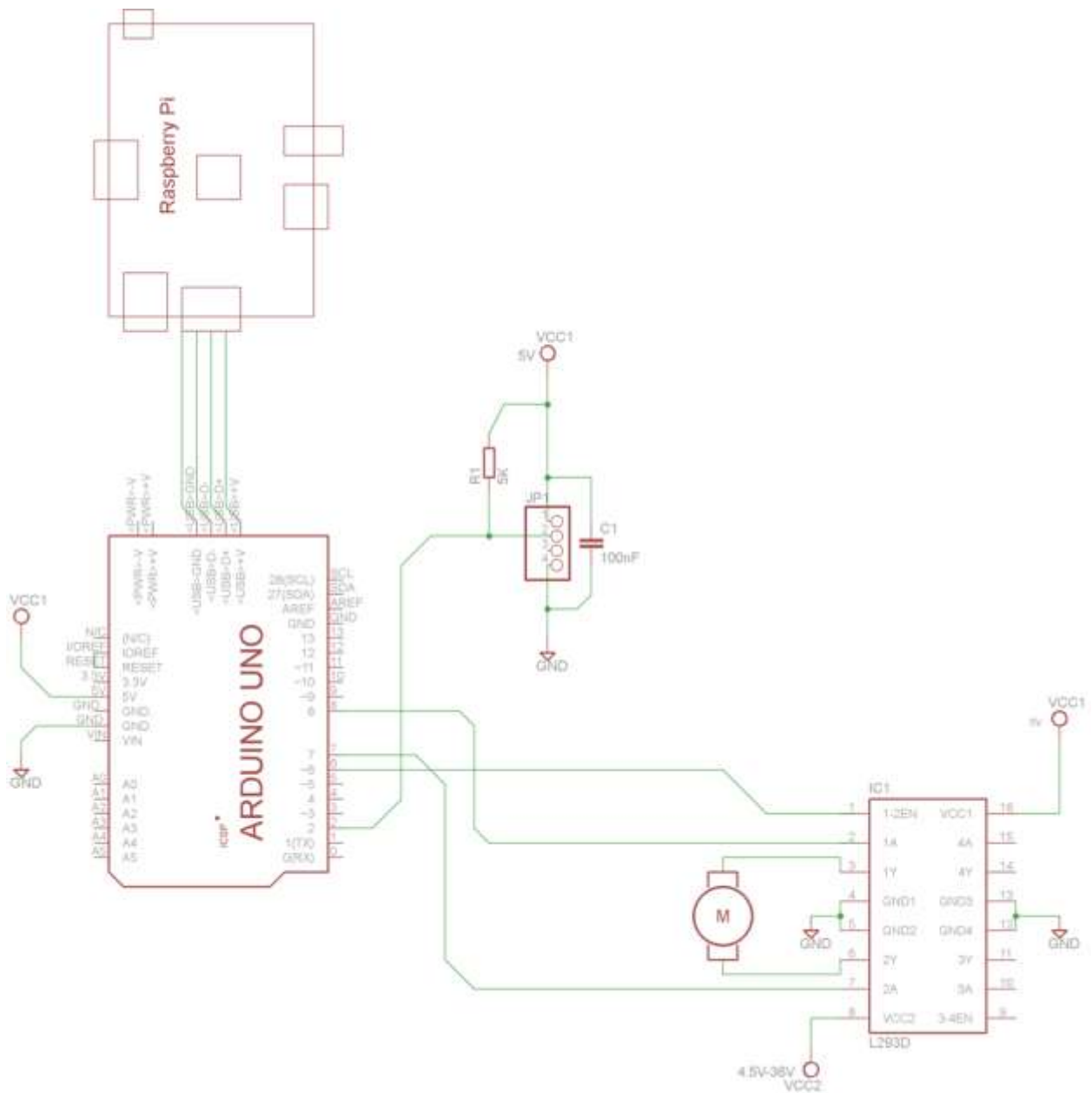


Figure 41. System schematic.

APPENDIX 2.

```
#!/usr/bin/python
#Description : The code allows the communication between Raspberry Pi and Arduino.
#              In addition, the data that are received, they will store to the SQLite
              database.
#Place : University of Vaasa
#Created by Antonios Plytas
#####
```

```
#!/usr/bin/python
import time,sched
import serial
import sqlite3
from time import localtime, strftime
```

```
##### FUNCTION FOR SENDING #####
```

```
def sendRecv(n):
    port.write(n)

    recv = port.read(25)

    try:
        print recv
        array = recv.split(" ")

        # Display data
        ident = int(array[0])
        humidity = int(array[1])
        temperature = int(array[2])
        dewPoint = int(array[3])
        checkSum = int(array[4])
        st = time.strftime('%Y-%m-%d %H:%M:%S')
        checkSumComp = ident + humidity + temperature + dewPoint
```

```

if checkSumComp == checkSum:
    humidity = float(humidity/100)
    temperature = float(temperature/100)
    dewPoint = float(dewPoint/100)
    writeToDatabase(ident, humidity, temperature, dewPoint, st)
else:
    print "checksum error"

```

```

except:
    print "Data not received correctly"
    port.write('r')
    port.flushInput()
    port.flushOutput()
    time.sleep(1)

```

WRITE TO DATABASE

```

def writeToDatabase(sensorID, dbHumidity, dbTemperature, dbDewpoint, dbTimestamp):

```

```

    #connect database
    conn = sqlite3.connect('test.db')
    print "Writing to database";
    conn.execute("INSERT INTO DHT11(sensorID, Humidity,
Temperature, Dewpoint, TIMESTAMP) \
    VALUES (?, ?, ?, ?, ?)", (sensorID, dbHumidity, dbTemperature,
dbDewpoint, dbTimestamp));

```

```

    #save changes
    conn.commit()
    print "Records created successfully";
    #close database
    conn.close()

```

TIMER

```

# time scheduler
s = sched.scheduler(time.time, time.sleep)

```

```
# serial port
port = serial.Serial("/dev/ttyACM0", baudrate=2400, timeout=12)

def timer_loop(sc):

    print "Read Sensor Data"

    sendRecv("t")

    sc.enter(15, 1, timer_loop, (sc,))

s.enter(15, 1, timer_loop, (s,))
s.run()
```

APPENDIX 3.

```

/*
 * Place: University of Vaasa
 * Description: Temperature, humidity and dew point measurements with motor speed
               control.
 * Board: Arduino Uno
 * Created by Antonios Plytas
 */

//Import libraries
#include <dht.h>
#include <avr/wdt.h>
#include <stdio.h>

##### Define constants #####
#define dht_dpin 2

dht DHT;
uint8_tstopM = 0;
uint8_tlastHumidity;

##### Setup #####
void setup(){

wdt_disable();
delay(200);
Serial.begin(2400);

##### Define Outputs #####
pinMode(7,OUTPUT);
pinMode(8,OUTPUT);
turnMotor(0,'s');
delay(1000);
}

```

```
##### Loop #####
```

```
void loop(){
DHT.read11(dht_dpin); //read the sensor

//Display measurements
char serialByte;
float humidity = 0, temperature =0, dewPoint = 0;
humidity = (float)(DHT.humidity);
temperature = (float)(DHT.temperature);
dewPoint = DewPoint(temperature,humidity);

if (Serial.available(>0) //read the serial port
{
serialByte = Serial.read();
if(serialByte == 't')
sendPacket( 1, humidity, temperature, dewPoint);
else if(serialByte == 'r')
softwareReboot(); //reset
}

fanSpeed((uint8_t) humidity);
delay(10000);
}
```

```
##### Fan Speed #####
```

```
void fanSpeed (uint8_t humidity)
{
if(lastHumidity == humidity)
return;

if(humidity < 34 )
{
analogWrite(6, 0); // pinEn requests a 0% duty cycle
```

```

}

else if(humidity > 34 && humidity <= 38)
{

turnMotor(127, 'r'); // pinEn requests a 50% duty cycle
//right direction

}

else if(humidity > 38 && humidity <= 45)
{
if(stopM == 0)
stopM = 1;
else if (stopM == 2)
{
turnMotor(0, 's');
delay(3000);
stopM = 1;
}

turnMotor(191, 'r'); // pinEn requests a 75% duty cycle
//right direction

}

else if(humidity > 45 )
{

if(stopM == 1){
turnMotor(0, 's');
delay(3000);
stopM = 2;
}
turnMotor(255, 'l'); // pinEn requests a 100% duty cycle
//left direction

}

```



```
lastHumidity = humidity;
}
```

```
##### Motor Direction Control #####
```

```
void turnMotor(const uint8_t dutyCycle, char direct)
{
  analogWrite(6, dutyCycle);
  switch(direct)
  {
    case 'r': digitalWrite(8,LOW); //right direction
              digitalWrite(7,HIGH);
              break;

    case 'l': digitalWrite(8,HIGH); //left direction
              digitalWrite(7,LOW);
              break;

    case 's': digitalWrite(8,LOW); //stop
              digitalWrite(7,LOW);
              break;
  }
}
```

```
##### String data packet #####
```

```
void sendPacket( const uint8_t id, const float humidity, const float temperature, const float
dewPoint)
{
  uint16_t temp, hum, dewP, checksum;
  char buffer[250];
  temp = (uint16_t)temperature * 100;
  hum = (uint16_t)humidity * 100;
  dewP = (uint16_t)dewPoint * 100;
  checksum = id + hum + temp + dewP;
  sprintf(buffer,"%hu %hu %hu %hu %hu", id, hum, temp, dewP, checksum);
```

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

```
##### Dew Point measurement #####
```

```
float DewPoint(float temperature, float humidity)
{ //reference: http://en.wikipedia.org/wiki/Dew\_point
float b = 17.67;
float c = 243.5;
float g = (b * temperature) / (c + temperature) + log(humidity/100);
floatTdp = (c * g) / (b - g);
returnTdp;
}
```

```
##### Reset #####
```

```
void softwareReboot()
{
wdt_enable(WDTO_15MS);
while(1)
{
}
}
```

APPENDIX 4.

The overall system during testing procedure.

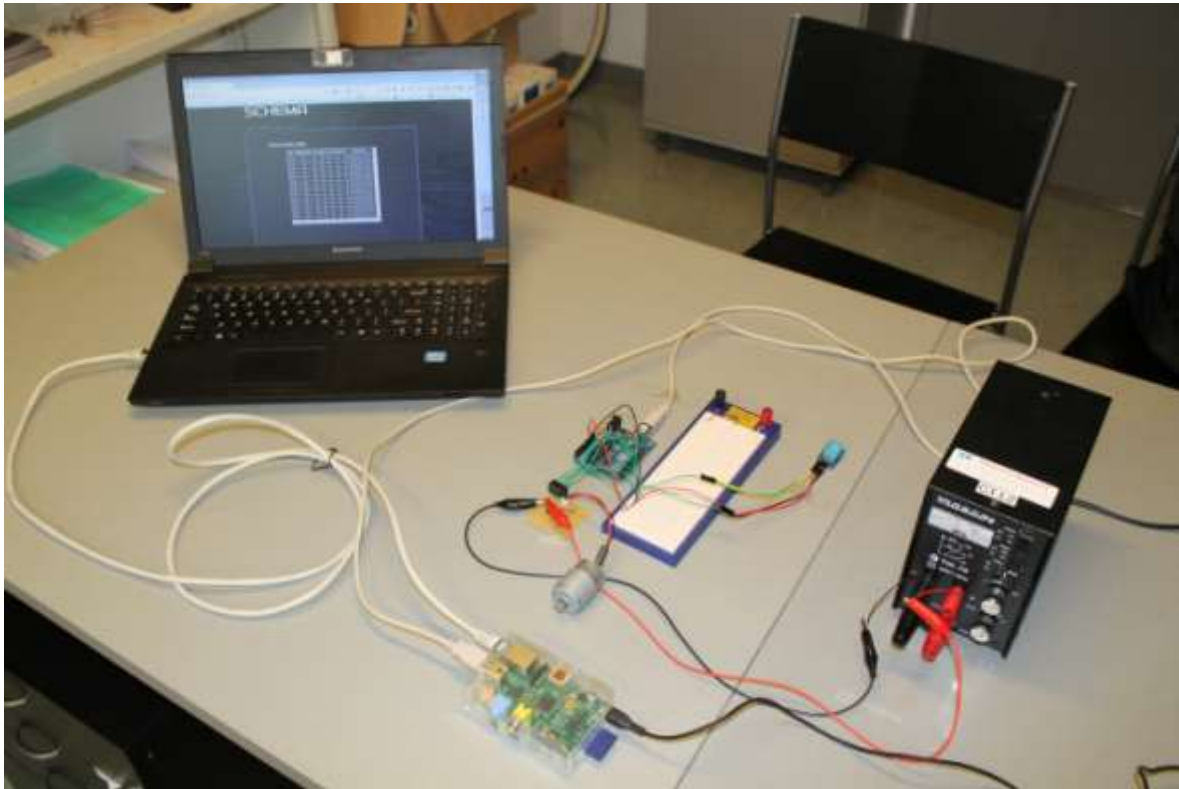


Figure 42. Overall system.