

CHAPTER 1

INTRODUCTION

1.1 SCOPE

After a lot of research in the field of technology, automation techniques became a reality. Still, they were quite expensive and found application in industrial fields where human effort was risky, inefficient or too slow. Automation today, has become a luxury and is soon going to find its place in every home. Home automation is not only about convenience, but also about better security. Initially, the home automation systems were built only for security purpose. Advances in home networking technology using both wired and wireless Ethernet have enabled the addition of more devices to the network at an affordable incremental cost.

Home automation system makes the operations of various home appliances more convenient and saves energy. With the energy saving concept, home automation or building automation makes life very simple nowadays. It involves automatic controlling of all electrical or electronic devices in homes or even remotely through wireless communication. Centralized control of lighting equipments and all other electronic equipments used in home systems is possible with this system. Wireless Home automation is the expansion and advancement of wired automation which uses wireless technologies like IR, Zigbee, Wi-Fi, GSM, Bluetooth, etc., for achieving remote operation. Here in this project Bluetooth wireless technology based home automation is implemented with energy conservation in mind.

1.2 OBJECTIVES

The aim of the project is to implement a low cost, reliable home automation system that can be used to control any household appliances, using a microcontroller to achieve hardware simplicity. The main objective of the system is to optimize the

electrical energy consumptions by the electronic devices or home appliances. This is achieved by automatically adjusting the electronic devices based on the environmental conditions such as temperature, luminance, etc of the home.

1.3 DOMAIN INTRODUCTION

1.3.1 Internet of Things

The Internet of Things (IoT) is the network of physical objects or "things" embedded with electronics, software, sensors, and connectivity to enable objects to exchange data with the production, operator or other connected devices. Internet of Things allows objects to be sensed and controlled remotely across existing network infrastructure, creating opportunities for more direct integration between the physical world and computer-based systems, and resulting in improved efficiency, accuracy and economic benefit. Each thing is uniquely identifiable through its embedded computing system but is able to interoperate within the existing Internet infrastructure.

Internet of Things provides everything needed to generate data driven intelligence from connected things, people and devices. These devices collect useful data with the help of various existing technologies. Typically, IoT is expected to offer advanced connectivity of devices, systems and services that goes beyond machine-to-machine (M2M) communications and covers a variety of protocols, domains, and applications. The interconnection of these embedded devices (including smart objects), is expected to be used in automation in nearly all fields, while also enabling advanced applications like a smart grid, and expanding to the areas such as smart cities.

"Things" in the IoT sense, can refer to a wide variety of devices. The expansion of Internet-connected automation into a plethora of new application areas, IoT is also expected to generate large amounts of data from diverse locations, with the

consequent necessity for quick aggregation of the data, and an increase in the need to index, store, and process such data more effectively. IoT is one of the platforms of today's Smart City, and Smart Energy Management Systems.

Home automation systems, like other building automation systems, are typically used to control lighting, heating, ventilation, air conditioning appliances, communication systems, entertainment and home security devices to improve convenience, comfort, energy efficiency and security.

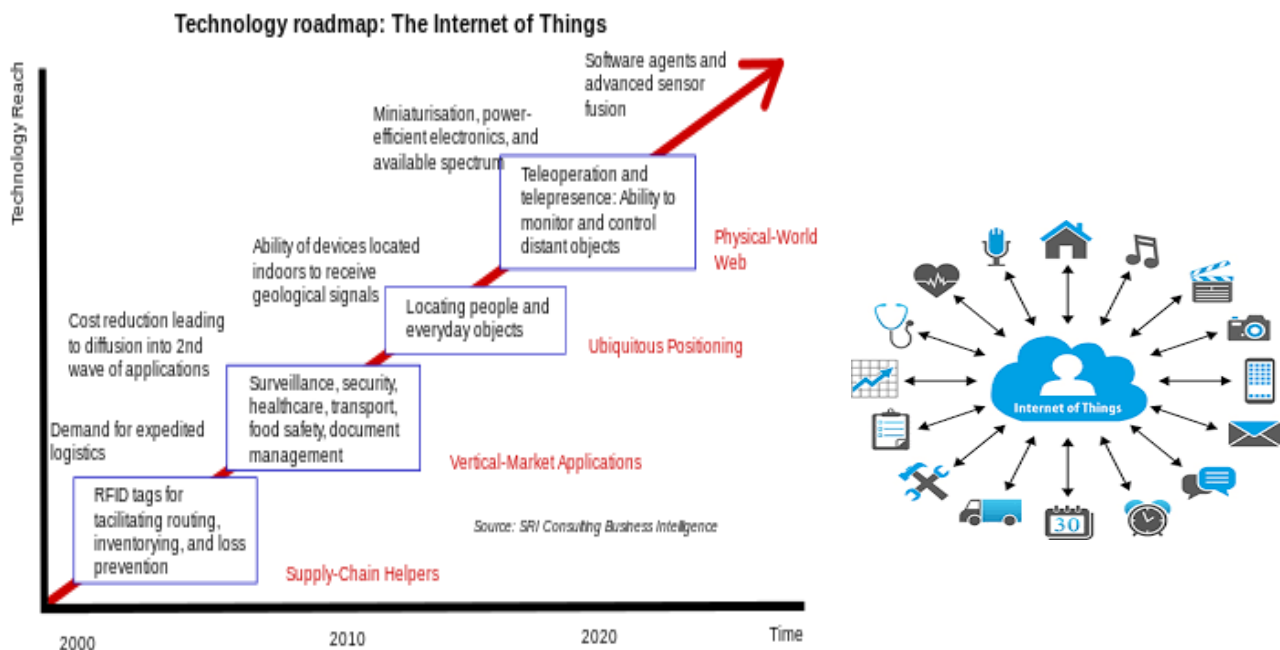


Fig 1.1 Technology roadmap of IOT

The above figure shows the technology roadmap and things connected to IoT. IoT has its applications in various fields such as Smart home, Wearable, Smart City, Smart grids, Industrial internet, connected car, Connected Health (Digital health/Tele-health/Tele-medicine), Smart retail, etc.

1.3.2 Wireless Personal Area Network

A wireless personal area network (WPAN) is a personal, short distance area wireless network for interconnecting devices centered on an individual person's

workspace. It covers an area of only a few dozen meters. WPANs address wireless networking and mobile computing devices such as PCs, PDAs, peripherals, cell phones, pagers and consumer electronics. Wireless PAN is based on the standard IEEE 802.15.

A WPAN could serve to interconnect all the ordinary computing and communicating devices that many people have on their desk or carry with them today; or it could serve a more specialized purpose such as allowing the surgeon and other team members to communicate during an operation. A key concept in WPAN technology is known as "plugging in". In the ideal scenario, when any two WPAN-equipped devices come into close proximity (within several meters of each other) or within a few kilometres of a central server, they can communicate as if connected by a cable. Another important feature is the ability of each device to lock out other devices selectively, preventing needless interference or unauthorized access to information.

The technology for WPANs is in its infancy and is undergoing rapid development. Proposed operating frequencies are around 2.4 GHz in digital modes. The objective is to facilitate seamless operation among home or business devices and systems. Every device in a WPAN will be able to plug into any other device in the same WPAN, provided they are within physical range of one another. In addition, WPANs worldwide will be interconnected. There are several kinds of technology used for WPANs:

1. The main WPAN technology is **Bluetooth**, which offers a maximum throughput of 1 Mbps over a maximum range of about thirty meters. For example, Bluetooth devices such as a keyboards, pointing devices, audio headsets, printers may connect to personal digital assistants (PDAs), cell phones, or computers wirelessly. A Bluetooth PAN is also called a piconet. It is composed of up to 8 active devices in a

master-slave relationship. It has the advantage of being very energy-efficient, which makes it particularly well suited to use in small devices.

Bluetooth operates at frequencies between 2402 and 2480 MHz, or 2400 and 2483.5 MHz including guard bands 2 MHz wide at the bottom end and 3.5 MHz wide at the top. This is in the globally unlicensed (but not unregulated) industrial, scientific and medical (ISM) 2.4 GHz short-range radio frequency band. Bluetooth uses a radio technology called frequency-hopping spread spectrum.

Bluetooth divides transmitted data into packets, and transmits each packet on one of 79 designated Bluetooth channels. Each channel has a bandwidth of 1 MHz. It usually performs 800 hops per second, with Adaptive Frequency-Hopping (AFH) enabled. The devices can switch roles, by agreement, and the slave can become the master. At any given time, data can be transferred between the master and one other device (except for the little-used broadcast mode). The master chooses which slave device to address; typically, it switches rapidly from one device to another in a round-robin fashion. Since it is the master that chooses which slave to address, whereas a slave is (in theory) supposed to listen in each receive slot, being a master is a lighter burden than being a slave. Being a master of seven slaves is possible; being a slave of more than one master is possible. Since Bluetooth WPAN is energy efficient and has several advantages over other networks Bluetooth wireless technology has been used in this system.

2. Home-RF has a maximum throughput of 10 Mbps with a range of about 50 to 100 meter without an amplifier. It is a wireless networking specification for home devices. It was developed in 1998 by the Home Radio Frequency Working Group, a consortium of mobile wireless companies that included Proxim Wireless, Intel, Siemens AG, Motorola, Philips and more than 100 other companies. The HomeRF standard, despite Intel's support, was abandoned, largely because processor

manufacturers had started to support on-board Wi-Fi (via Centrino technology, which included a microprocessor and a Wi-Fi adapter on a single component).

3. The technology **ZigBee** (also known as IEEE 802.15.4) can be used to connect devices wirelessly at a very low cost, which makes it particularly well-suited for being directly integrated into some small electronic appliances (like home appliances, stereos and toys). ZigBee, which operates on the frequency band of 2.4 GHz and on 16 channels, can reach transfer speeds of up to 250 Kbps with a maximum range of about 100 meter. It is the wireless language that everyday devices use to connect to one another.

4. Finally, **infrared** connections can be used to create wireless connections over a few meters, with speeds that can reach a few megabits per second. This technology is widely used in home electronics (like remote controls), but light waves can interfere with the signal. It plays a significant role in Wireless Personal Area Networks (WPANs). Its key features (including short range, narrow cone, dynamic ad hoc connectivity, rapid connection establishment, high data rates, low power, and low cost) make it an ideal technology for certain usage models. In other situations, its ability to co-exist with, and complement Bluetooth solutions also make it an important player in this space.

1.3.3 Home Automation Components

Automation is one of the two main characteristics of home automation. Automation refers to the ability to program and schedule events for the devices on the network. The programming may include time-related commands, such as having one's lights turn on or off at specific times each day.

Home automation most commonly connects simple binary devices. This includes "on and off" devices such as lights, power outlets and electronic locks, but also devices such as security sensors which have only two states, open and closed.

The classic control unit is the home computer, for which many of the earlier home automation systems were designed. Today's home automation systems are more likely to distribute programming and monitoring control between a dedicated device in the home, like the control panel of a security system, and a user-friendly app interface that can be accessed via an Internet-enabled PC, smartphone or tablet.

CHAPTER 2

LITERATURE SURVEY

2.1 A Review of Smart home-Past, Present and Future. Muhammad RaisulSalam, University of Toronto, Mamun Bin IbneReaz, University of Kebangsaan Malaysia.

Muhammad RaisulSalam had given an idea of various methodologies algorithms, communication protocols that can be incorporated in developing this Smart home [1]. Ideas were provided for security, energy conservation, health monitoring. They also tell about different types of algorithms for predicting human activities and methodologies, different communication protocols that can be incorporated for developing Smart home.

Their work presents a general overview of smart home projects that are arranged according to their intended services. They also discussed the significance and limitations of smart home building blocks. The taxonomy of devices, media, protocols, algorithms, methods, and services presents an informative comparison between the associated technologies. Their system would also use different user interfaces to acquire user feedback, most of which would be based on auditory, visual, and haptic perceptions.

2.2 Internet of Things based Energy Aware Smart Home Control System,Murad Khan, BhagyaNathali Silva, Kijun Han.

A Home Energy Management (HEM) system for power consumption control based on a predefined voltage threshold during peak hours has been explained by Murad Khan et al [2]. It also deals with smart interference control system, smart energy control system and smart management control system. Home Management System follows a customer experience model to achieve high customer comfort level.

A scheduling technique of electrical and thermal appliances based on resident behavior, weather conditions, discomfort index, etc. The energy of various renewable energy sources is computed and incorporated it for operating various home appliances such as a boiler, vacuum cleaner.

2.3 Design of Smart Home System Based on ZigBee Technology and R&D for Application, Lin Gao¹, Zhixin Wang¹, Jianlong Zhou², Chao Zhang³

Lin Gao¹ et al [3] proposed the smart home system based on ZigBee technology uses wireless communications technology, and can realize the collection of the electric energy information of home devices through the electric energy metering chip integrated in the smart socket, and realize the remote control and data monitoring of each device through the mobile terminal. Meanwhile, the home server can store electric energy information and electricity consumption of home devices in the background database, and communicates with the mobile terminal by Socket of TCP protocol.

2.4 WSN-Based Smart Sensors and Actuator for Power Management in Intelligent Buildings, Nagender Kumar Suryadevara, Student Member, IEEE, Subhas Chandra Mukhopadhyay, Fellow, IEEE, Sean Dieter, Tebje Kelly, and Satinder Pal Singh Gill.

Nagender Kumar Suryadevara et al [4] focused on human-friendly technical solutions for monitoring and easy control of household appliances. This paper aims determine the areas of daily peak hours of electricity usage levels and come with a solution by which we can lower the consumption and enhance better utilization of already limited resources during peak hours. By analyzing the power from the system, energy consumption can be controlled. An electricity tariff plan has been set up to run various appliances at peak and off-peak tariff rates.

2.5 Human Voice Controlled Home Appliances, prateekGupta,Student, Electronics and Telecommunication Engineering, BharatiVidyapeeth University C.O.E, Pune, India

The author Prateek Gupta [5] presented a Practical voice recognition kit which was utilized in order to store and recognize the user's voice. AVR micro controller has been used since it is best when interfaced with voice recognition technology. About 5% errors occurred here, this was because the recognizer may hear a different pronunciation.

2.6 Voice Control of Home Appliances using Android, NorhafizahbtAripin and M. B. Othman

Norhafizah bt Aripin and M. B. Othman [6] presented the development of home appliances based on voice command using Android. Google application has been used as voice recognition and process the voice input from the smart phone. The voice input has been captured by the android and will be sent to the Arduino Uno. Bluetooth module in Arduino Uno received the signal and processed the input signal to control the light and fan.

2.7 IoT Based Home Automation and Surveillance System, Syed Ali Imran Quadri,P.Sathish,Department of Electronics and Communication Engineering,ChaitanyaBharathi Institute of Technology, Gandipet, Hyderabad, India.

Syed Ali Imran Quadri, P.Sathish [7] discussed on security issues that occur in home namely intrusion, theft, fire and leakage of flammable gas which can be identified through a webcam that has been connected to one of the USB ports of the Raspberry pi. The live streaming of the video can be viewed once the user correctly enters the username and password in the index.html page that opens upon startup of the Raspberry pi.

2.8 Bluetooth based home automation system, N. Sriskanthan*, F. Tan, A. Karande, School of Computer Engineering, Nanyang Technological University, Nanyang Avenue, Singapore, Singapore.

N. Sriskanthan, F. Tan, A. Karande[8] connected Various appliances via bluetooth and monitored using microcontroller. On connectivity with bluetooth these devices do not act as an individual one, rather they form a personal network in which these appliances communicate with each other. Various protocols has been used for communication within this this personal network.

2.9 Smart Living Using Bluetooth Based Android Smartphone, Ming Yan and Hao Shi College of Engineering and Science, VictoriaUniversity, Melbourne, Australia.

Ming Yan and Hao Shi [9] proposed Home lighting control system which involves development of a small "piconet" using a microchip and several Bluetooth modules. Android application is developed to monitor the status of various lightings and a master slave structure is adopted where the android application enabled with Bluetooth acts as the master while other Bluetooth devices, for this instance, switches, linked to the home lighting system are slave devices.

CHAPTER 3

PROPOSED SYSTEM

3.1 SYSTEM ARCHITECTURE

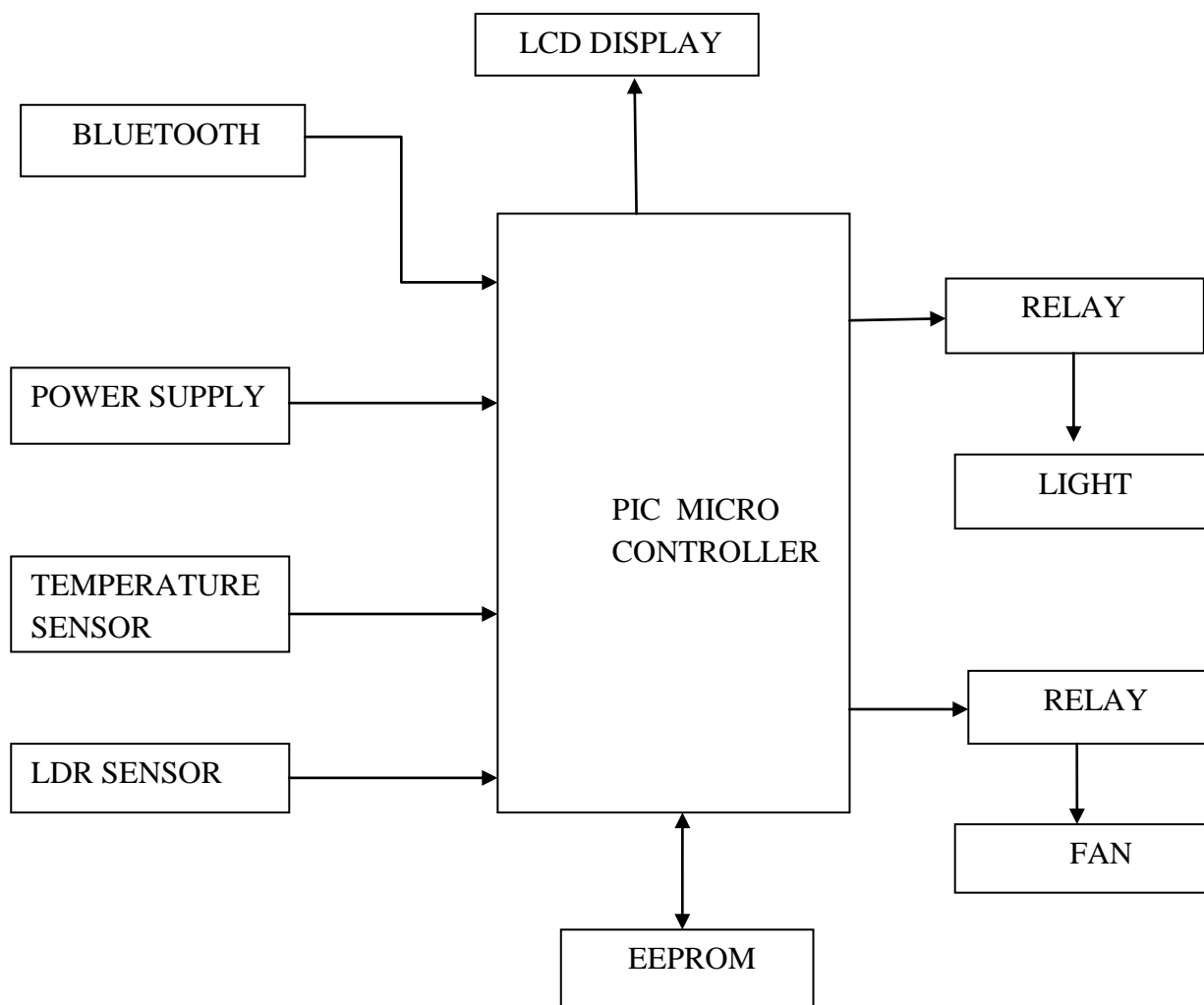


Fig 3.1 System Architecture

The above architecture depicts the connectivity of hardware components. Microcontroller has been interfaced with various components such as LED (Light Emitting Diode), LCD (Liquid Crystal Display), temperature sensor, LDR (Light Dependent Resistor) and a Bluetooth module.

3.2 PROPOSED METHDOLOGY

A smart home solution with energy conservation in mind has been presented. Two home appliances that have been chosen for this project are fan and light. An android application is built to help user to access the system through voice commands. Google application has been used as voice recognition and process the voice input from the android application in smart phone. A PIC microcontroller and a separate arduino processor are used for processing data and controlling appliances. The Bluetooth module in microcontroller receives the input and process the input signal to control the light and fan. The light intensity and the fan speed can be adjusted according to the comfort level or controlled according to the occupant's activity based needs.

3.2.1 PROJECT OVERVIEW

The system developed is classified into 2 phases. 1. User Interface 2.Hardware. Android application here acts as a user interface via which the user can interact with the system by using voice commands in mobile device. The hardware phase includes the hardware components used to build the system. The system provides two different modes of operations. They are

1) Auto Mode 2) Normal Mode

To operate in such modes synchronization of hardware and software is very important. This is achieved by pairing the mobile device and the microcontroller via Bluetooth through android application. Once the pairing is done, android application accepts voice commands and allows the user to select the modes, so that the system acts accordingly.

If the user choose normal mode of operation, the user is requested to provide the required speed of the fan and the luminance level of light in order to control the

appliances. PIC microcontroller is used to control the home appliance according to the voice command received from the user. PIC microcontroller controls the ON/OFF of the light and the fan. It also controls the luminance of the light and speed of fan.

If the user choose auto mode of operation, then the devices will be controlled automatically based on the atmospheric conditions. This is in turn PIC microcontroller collects information about external temperature and luminosity from temperature sensor and LDR (Light Dependent Resistor) and controls the speed and luminance of light and fan automatically.

Two different sensors namely LDR, temperature sensor has been used. LDR is used for measuring the intensity of light whereas temperature sensor is used for measuring temperature of the surroundings.

All the above mentioned sensors are connected to the PIC microcontroller which in turn is connected to LCD that displays status of the hardware.

3.2.2 MODULE DESCRIPTION:

1. Hardware interfacing
2. Device pairing
3. Mode selection
4. Device Control

1) HARDWARE INTERFACING

Two home appliances that have been chosen for this project are fan and light. Here we use a DC motor and LED to demonstrate the project where DC motor acts as fan and LED acts a light.

i)Interfacing DC motor with microcontroller

A DC Motor cannot be directly interfaced with the microcontroller, as DC Motors requires high current and high voltage than a Microcontroller can handle. Microcontrollers usually operates at +5 or +3.3V supply and its I/O pin can provide only up to 25mA current. Commonly used DC Motors requires 12V supply and 300mA current; moreover interfacing DC Motors directly with Microcontrollers may affect the working of Microcontroller due to the Back EMF of the DC Motor.

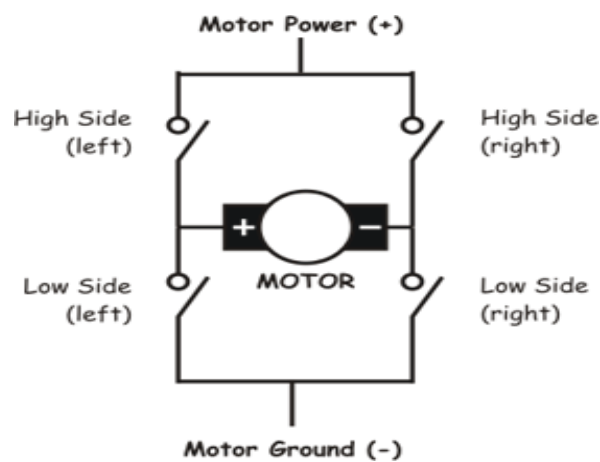


Fig 3.2 DC Motor Interfacing

It is a special circuit; by using the 4 switches the direction of DC Motor can be controlled. Depending upon our power requirements we can make our own H-bridge using Transistors/MOSFET's as switches. It is better to use readymade ICs, instead of making our own H-bridge.

L293D and L293 are two such ICs. These are dual H-bridge motor drivers, i.e. by using one IC two DC Motors can be controlled in both clock wise and counter clockwise directions. In this project L293D IC is used. The L293D can provide bidirectional drive currents of up to 600-mA at voltages from 4.5 V to 36 V.

When the user gives their command as fan on, fan off or fan low the microcontroller instructs the arduino processor to control the DC motor using motor driver.

ii) Interfacing LED with microcontroller

Fig.3.3 shows how to interface the LED to microcontroller. Anode is connected through a resistor to GND & the Cathode is connected to the Microcontroller pin. So when the Port Pin is HIGH the LED is OFF & when the Port Pin is LOW the LED is turned ON.

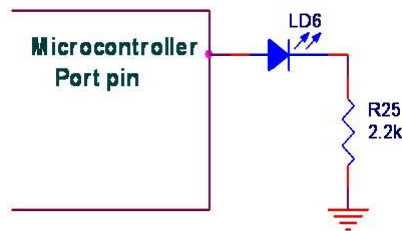


Fig.3.3 Interfacing LED to Microcontroller

It works by turning ON a LED & then turning it OFF & then looping back to START. However the operating speed of microcontroller is very high so the flashing frequency will also be very fast to be detected by human eye.

When the user gives their command as light on, light off or light low the microcontroller instructs the arduino processor to control the LED.

iii) Interfacing Sensors with Microcontroller

In this system, two sensors that are used are Temperature sensor and a LDR sensor. To collect the data from the sensors they need to be interfaced with the microcontroller. For interfacing, the sensors need to be connected with the microcontroller and PIC compiler need to be instructed to what to do with the sensors by coding the PIC compiler using the MPLABX software. The compiler will generate

a hex file for the uploaded code and then it is ready to collect information from the sensors.

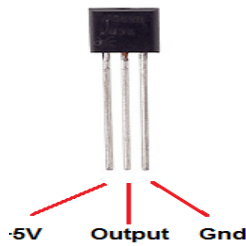


Fig 3.4 Temperature sensor

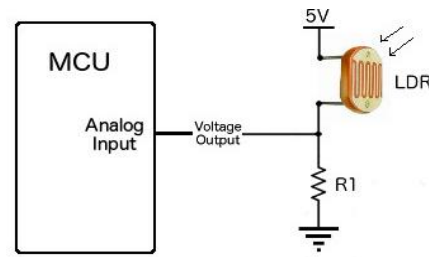


Fig 3.5 LDR sensor

The above two figure shows the sensors (temperature sensor and LDR sensor) that were interfaced with the microcontroller. Along with the above interfaced components LCD and Bluetooth module are also interfaced with the microcontroller.

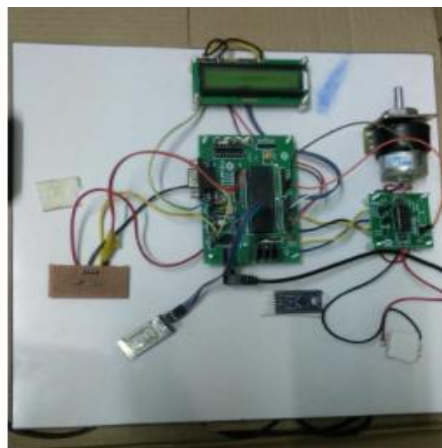


Fig 3.6 Hardware setup after interfacing

2) DEVICE PAIRING

The user can communicate with the controller to control the devices by sending voice commands from android application. To do so android device must be connected with the microcontroller via any wireless communication. Here Bluetooth Wireless communication is chosen, since it consumes less electrical energy and it is cost effective. Normally Bluetooth wireless technology is used daily for sending and receiving data from one device to another in mobile phones or computers. So it is

easier to work with. The block diagram of interfacing Bluetooth module with microcontroller is shown below.

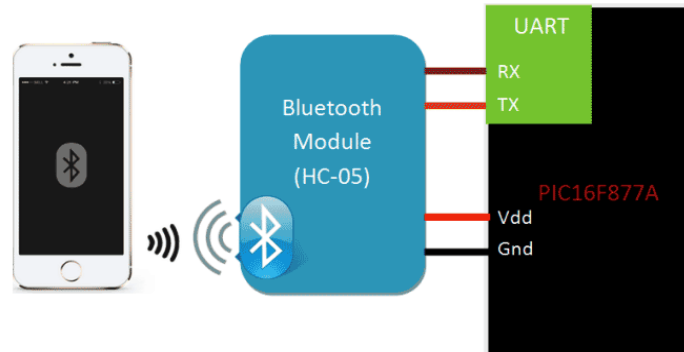


Fig 3.7 Device Pairing

Bluetooth modules need communication protocols to interface with other devices. For example, while interfacing Bluetooth module with microcontroller, microcontroller can communicate with Bluetooth device through following wired communications to send or receive data from other Bluetooth device:

- UART (Universal Asynchronous Receiver Transmitter)
- SPI (Serial Peripheral Interface)
- USB (Universal Serial Bus)

Here UART communication port is used. Once the Bluetooth in the android device and the Bluetooth module in the microcontroller are paired the LCD screen in the prototype displays the message VOICE CONTROL BASED HOME AUTOMATION. Now the user can start sending commands from the android application.



Fig 3.8 Initial status after pairing

3) MODE SELECTION

Once the device pairing is done, the user can start communicating with the system. To communicate, the user needs to visit the android application which is developed. The android application can accept voice commands as input. Now the user can select the mode of operation by sending the voice commands as NORMAL MODE or AUTO MODE.

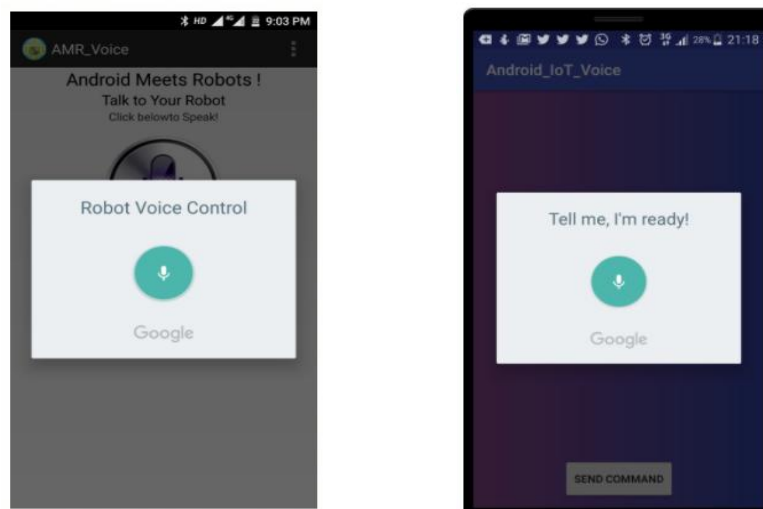


Fig 3.9 Sending Commands through android app

When the user gives command as NORMAL MODE, the android application asks for the speed of the fan and intensity of the light which is comfortable for them.

For example to turn on the fan at lower speed, 'speed 1' has been set in android with character 'A', 'speed 2' for character 'B' and etc. When the user touch the 'speed 1' button, the character 'A' will be sent to the microcontroller and the output of 'speed 1' will be turn on. The speed can be adjustable in 3 ways. The intensity of the light can be adjusted accordingly from 0 to 1000.

When the user gives command as AUTO MODE, the devices will be controlled automatically. Now it is the job of the sensors and the microcontroller to control the device. In normal mode, the temperature sensor and LDR sensors are activated to collect the data from the surrounding. The collected data will be send to the microcontroller and microcontroller in turn controls the device.

4) DEVICE CONTROL

The Bluetooth module in the microcontroller receives the data sent from the android application. The processor in the microcontroller does the work of receiving and forwarding the input signal. The processor in the microcontroller forwards the input signal to the arduino processor. The arduino processor in turn is used to control the appliances. The microcontroller processes the received input and operates accordingly. For the normal mode of operation the microcontroller controls the fan and light with respect to the inputs given by the user. Thus the speed of the fan and intensity of the light are controlled. For auto mode of operation, the microcontroller collects information such as temperature and luminance from temperature sensor and LDR. With the collected data the controller controls the devices.

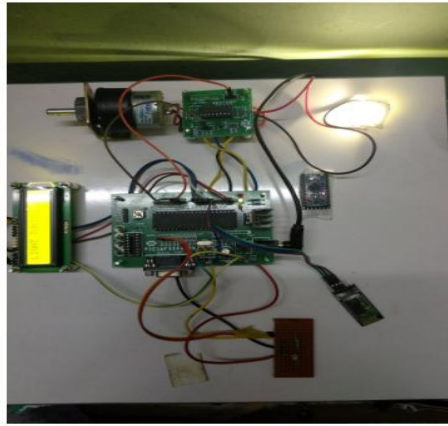


Fig3.10 light turning on through voice command

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

- ☐ PIC Microcontroller
- ☐ Arduino Processor
- ☐ Temperature sensor
- ☐ Light Dependent Resistor
- ☐ Liquid Crystal Display
- ☐ Light emitting Diode
- ☐ Motor
- ☐ Motor Driver

4.2 SOFTWARE REQUIREMENTS

Operating system: windows 8

IDE USED: Eclipse, MP LABX

CHAPTER 5

CONCLUSION AND FUTURE WORKS

5.1 CONCLUSION

Thus the Voice control based home automation has been successfully built and developed. Bluetooth wireless Technology has been used to control home appliance. An android application is built to help user to access the system through voice commands. It can be implemented in organisation where there is a need to optimize the use of electricity in a smart way. Since voice is given as input there may be chances of disruption or disturbances which may lead to improper voice command that might not be recognized, so external disturbances must be avoided while giving this voice command as input.

5.2 FUTURE WORKS

Future homes will be able to offer almost all required services e.g., communication, medical, energy, utility, entertainment and security. People spend a significant amount of time in their houses, which attracts potential investors to promote the integration of all possible services into traditional homes. Current trends in smart home research imply that health care services will receive more emphasis in the future. One of the main objectives will be providing assistive services for elderly and disabled. Remote patient monitoring will become more popular because providing healthcare services to certain groups of patients requires less manpower. Other services related to comfort and security will be improved gradually with the improvement of associated components.

APPENDIX-1

HARDWARE DESCRIPTION

PIC PROGRAMMABLE MICROCONTROLLER (16F877A)

PIC stands for Peripheral Interface Controller. A typical microcontroller includes a processor, memory, and peripherals. Every PIC microcontroller has a set of registers that also function as RAM (random access memory). Special purpose control registers for on-chip hardware resources are also mapped into the data space. Every PIC has a stack that saves return addresses. The stack was not software-accessible on the earlier versions of the PIC, but this limitation was removed in later devices. These programming and the simulated process of this microcontroller can be done by a circuit-wizard software. PIC microcontroller is an IC and its architecture comprises of CPU, RAM, ROM, timers, counters and protocols like SPI, UART which are used for interfacing with other peripherals. Applications of microcontroller include industrial purpose. The advantages of using this microcontroller include low power consumption, high performance, supports hardware and software tools such as simulators, compilers, and debuggers.

The high performance of the PIC micro devices can be attributed to a number of architectural features commonly found in RISC microprocessors. These include Harvard architecture, Long Word Instructions, Single Word Instructions, Single Cycle Instructions, Instruction Pipelining, Reduced Instruction Set, Register File Architecture, Orthogonal (Symmetric) Instructions. General purpose I/O pins can be considered the simplest of peripherals. The direction of the I/O pins (input or output) is controlled by the data direction register, called the TRIS register. TRIS<x> controls the direction of PORT<x>. A '1' in the TRIS bit corresponds to that pin being an input, while a '0' corresponds to that pin being an output. To configure the pins as a digital port, the corresponding bits in the LCDSE register must be cleared.

Any bit set in the LCDSE register overrides any bit settings in the corresponding TRIS register.

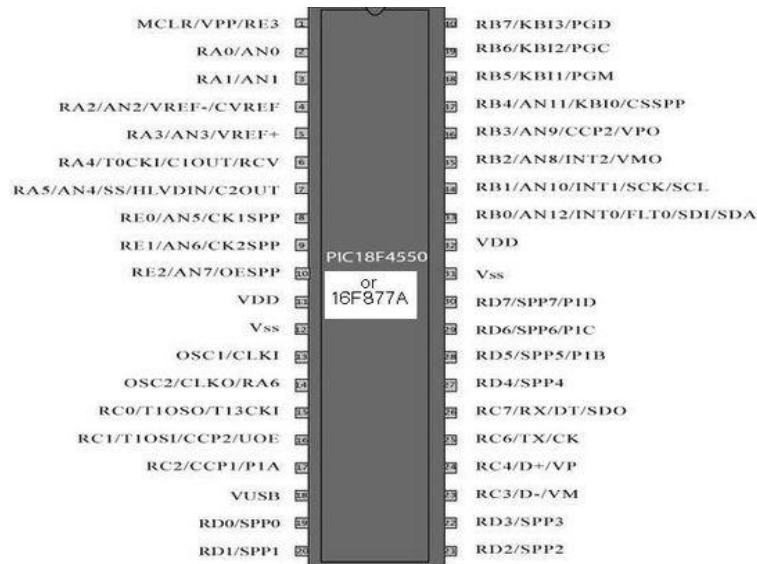


Fig A1.1 Pin out diagram of PIC

LIGHT DIODE RESISTOR

A photo-resistor or light-dependent resistor (LDR) or photocell is a light-controlled variable resistor. The resistance of a photo-resistor decreases with increasing incident light intensity; in other words, it exhibits photoconductivity. A photo-resistor can be applied in light-sensitive detector circuits, and light- and dark-activated switching circuits.

A photo-resistor is made of a high resistance semiconductor. In the dark, a photo-resistor can have a resistance as high as a few megaohms (MΩ), while in the light, a photo-resistor can have a resistance as low as a few hundred ohms. If incident light on a photo-resistor exceeds a certain frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electrons conduct electricity, thereby lowering resistance. The

resistance range and sensitivity of a photo-resistor can substantially differ among dissimilar devices. Moreover, unique photoresistors may react substantially differently to photons within certain wavelength bands.

The figure below shows that when the torch is turned on, the resistance of the LDR falls, allowing current to pass through it. When a light level of 1000 lux (bright light) is directed towards it, the resistance is 400R (ohms). When a light level of 10 lux (very low light level) is directed towards it, the resistance has risen dramatically to 10.43M (10430000 ohms). This is an example of a light sensor circuit:

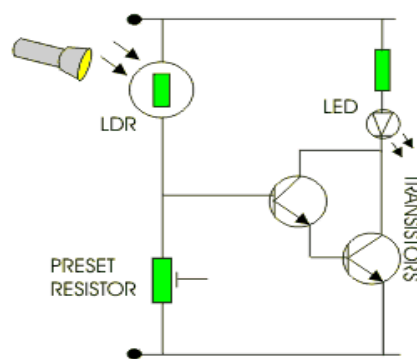


Fig A1.2 Working of LDR

When the light level is low the resistance of the LDR is high. This prevents current from flowing to the base of the transistors. Consequently the LED does not light. However, when light shines onto the LDR its resistance falls and current flows into the base of the first transistor and then the second transistor.

POWER SUPPLY

Almost all basic household electronic circuits need an unregulated AC to be converted to constant DC, in order to operate the electronic device. All devices will have a certain power supply limit and the electronic circuits inside these devices must be able to supply a constant DC voltage within this limit. That is, all the active and passive electronic devices will have a certain DC operating point (Q-point or

Quiescent point), and this point must be achieved by the source of DC power. The DC power supply is practically converted to each and every stage in an electronic system. Thus a common requirement for all these phases will be the DC power supply. All low power systems can be run with a battery. But, for long time operating devices, batteries could prove to be costly and complicated. The best method used is in the form of an unregulated power supply – a combination of a transformer, rectifier and a filter. The diagram is shown below.

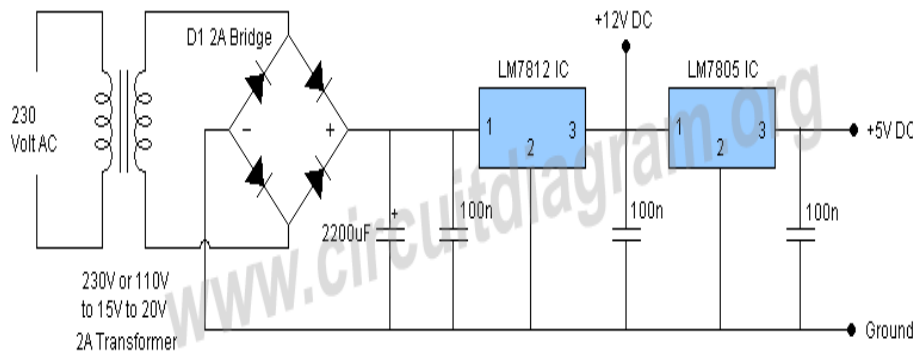


Fig A1.3 Step down transformer

As shown in the figure A1.3, a small step down transformer is used to reduce the voltage level to the device's needs. In India, a 1 ϕ supply is available at 230 volts. The output of the transformer is a pulsating sinusoidal AC voltage, which is converted to pulsating DC with the help of a rectifier. This output is given to a filter circuit which reduces the AC ripples, and passes the DC components. But there are certain disadvantages in using an unregulated power supply.

1. Poor Regulation – When the load varies, the output does not appear constant. The output voltage changes by a great value due to the huge change in current drawn from the supply. This is mainly due to the high internal resistance of the power supply (>30 Ohms).

2. AC Supply Main Variations – The maximum variations in AC supply mains is give or take 6% of its rated value . But this value may go higher in some countries (180-280 volts). When the value is higher it's DC voltage output will differ largely.

3. Temperature Variation – The use of semiconductor devices in electronic devices may cause variation in temperature.

These variations in dc output voltage may cause inaccurate or erratic operation or even malfunctioning of many electronic circuits. For instance, in oscillators the frequency will shift, in transmitters output will get distorted, and in amplifiers the operating point will shift causing bias instability. All the above listed problems are overcome with the help of a voltage regulator which is employed in conjunction with an unregulated power supply. Thus, the ripple voltage is largely reduced. Thus, the supply becomes a regulated power supply.

The internal circuitry of a regulated power supply also contains certain current limiting circuits which helps the supply circuit from getting fried from inadvertent circuits. Nowadays, all the power supplies use IC's to reduce ripples, enhance voltage regulation and for widened control options. Programmable power supplies are also available to allow remote operation that is useful in many settings.

VOLTAGE REGULATOR

Voltage sources in a circuit may have fluctuations resulting in not giving fixed voltage outputs. Voltage regulator IC maintains the output voltage at a constant value. 7805 IC, a voltage regulator integrated circuit (IC) is a member of 78xx series of fixed linear voltage regulator ICs used to maintain such fluctuations. The xx in 78xx indicates the fixed output voltage it provides. 7805 IC provides +5 volts regulated power supply in DC.

TEMPERATURE SENSOR

LM34 series of temperature sensors has been used. The LM34 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Fahrenheit temperature. The LM34 thus has an advantage over linear temperature sensors calibrated in degrees Kelvin, as the user is not required to subtract a large constant voltage from its output to obtain convenient Fahrenheit scaling. The LM34 does not require any external calibration or trimming to provide typical accuracies of $\pm 1.2^{\circ}\text{F}$ at room temperature and $\pm 11.2^{\circ}\text{F}$ over a full -50 to $+300^{\circ}\text{F}$ temperature range. The LM34 is rated to operate over a -50° to $+300^{\circ}\text{F}$ temperature range.

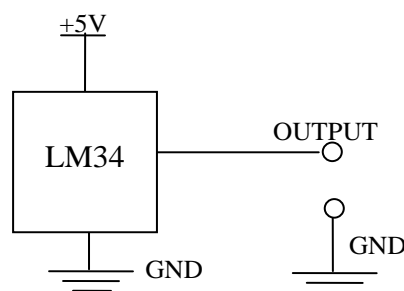


Fig A1.4 Circuit diagram for the LM34 temperature sensor functional module

It is easy to include the LM34 series in a temperature measuring application. The output voltage of LM34 is linearly proportional to the Fahrenheit temperature, it has a Linear $+10.0 \text{ mV}/^{\circ}\text{F}$ scale factor which means that it will produce $n \times 10.0 \text{ mV}$ output voltage if the environment temperature is $n^{\circ}\text{F}$.

The LM34 series is available packaged in hermetic TO-46 transistor packages, while the LM34C, LM34CA and LM34D are also available in the plastic TO-92 transistor package. The LM34D is also available in an 8-lead surface mount small outline package. In this functional module, LM34H in metal can package (TO-46) is used in the functional module, it is very important to know that the wiring of sensor should be based on the positions of the leading pins in different packages.

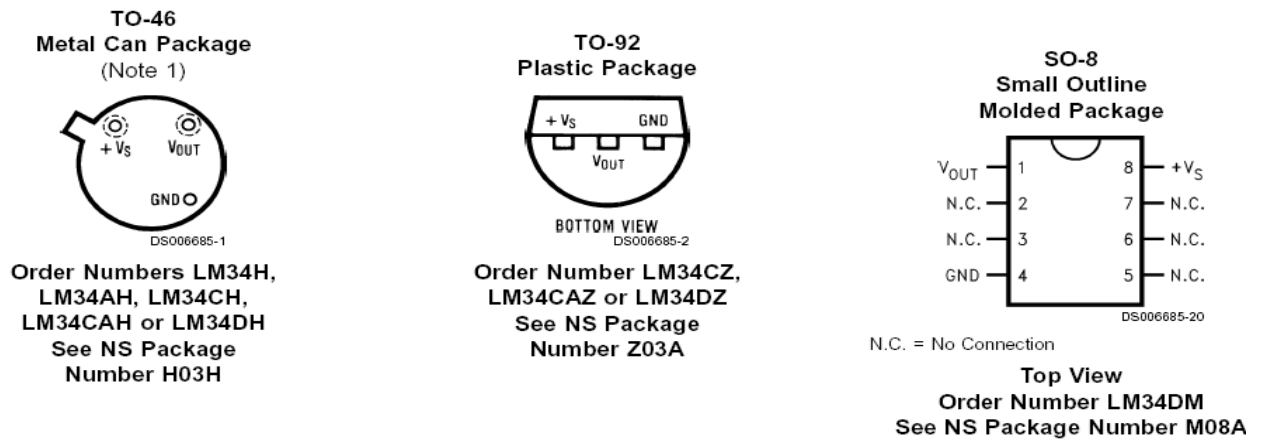


Fig.A1.5 Package Diagram of LM34

DESCRIPTION OF TEMPERATURE SENSOR FUNCTIONAL MODULE

The temperature sensor functional module consists of two parts: the function module box and the probe head. The LM34 temperature sensor is mounted on the probe head. Be careful to make sure that the sensor is properly mounted on the probe head.

By replacing the LM34 with another precision integrated-circuit temperature sensor LM35, it is easy to get an output voltage proportional to the centigrade temperature. The LM35 sensor has a linear $+10.0 \text{ mV}/^\circ\text{C}$ scale factor and a temperature range from -55°C to $+150^\circ\text{C}$. In fact LM34 and LM35 are among the same series of temperature sensors so that they can be easily exchanged in different applications. The wiring for LM 35 is the same as that of LM34.

EEPROM

EEPROM is non volatile memory so that it is uses for storing information for long life if power is loss. When power is activated that time only can change the data I side of the IC. This is used for generating alphabet letter for random purpose.

LCD

There are many display devices used by the hobbyists. LCD displays are one of the most sophisticated display devices used by them. Once the interfacing is done, it

will be the easiest and very reliable output device. Moreover, for micro controller based applications, not every time any debugger can be used. So LCD displays can be used to test the outputs.

LCD accepts two types of signals, one is data, and another is control. These signals are recognized by the LCD module from status of the RS pin. Now data can be read also from the LCD display, by pulling the R/W pin high. As soon as the E pin is pulsed, LCD display reads data at the falling edge of the pulse and executes it, same for the case of transmission.

LCD display takes a time of 39-43 μ S to place a character or execute a command. Except for clearing display and to seek cursor to home position it takes 1.53ms to 1.64ms. Any attempt to send any data before this interval may lead to failure to read data or execution of the current data in some devices. Some devices compensate the speed by storing the incoming data to some temporary registers.

LCD displays have two RAMs, naming DDRAM and CGRAM. DDRAM registers in which position which character in the ASCII chart would be displayed. Each byte of DDRAM represents each unique position on the LCD display. The LCD controller reads the information from the DDRAM and displays it on the LCD screen. CGRAM allows user to define their custom characters. For that purpose, address space for first 16 ASCII characters are reserved for users. After CGRAM has been setup to display characters, custom characters can be displayed on the LCD screen.

RELAY

A relay is an electromagnetic switch operated by a relatively small electric current that can turn on or off a much larger electric current. The heart of a relay is an electromagnet (a coil of wire that becomes a temporary magnet when electricity flows through it). General purpose relays operate with AC or DC current, at common

voltages such as 12V, 24V, 48V, 120V and 230V, and they can control currents ranging from 2A-30A. Here we are using 12v relay.

BLUETOOTH MODULE

The Bluetooth module HC-05 is a MASTER/SLAVE module. By default the factory setting is SLAVE. The Role of the module (Master or Slave) can be configured only by AT COMMANDS. The slave modules cannot initiate a connection to another Bluetooth device, but can accept connections. Master module can initiate a connection to other devices. The user can use it simply for a serial port replacement to establish connection between MCU and GPS, PC to embedded systems, etc.

APPENDIX 2

SOFTWARE DESCRIPTION

MPLAB



Fig A2.1 MPLAB Logo

MPLAB is a free integrated development environment for the development of embedded applications on PIC and dsPIC microcontrollers, and is developed by Microchip Technology.

MPLAB X is the latest edition of MPLAB, and is developed on the NetBeans platform. MPLAB and MPLAB X support project management, code editing, debugging and programming of Microchip 8-bit, 16-bit and 32-bit PIC microcontrollers.

MPLAB is designed to work with MPLAB-certified devices such as the MPLAB ICD 3 and MPLAB REAL ICE, for programming and debugging PIC microcontrollers using a personal.

MPLAB SETUP

MPLAB is Microchips Integrated Development Environment. It can be downloading directly from Microchips MLAPB Site. While using CCS' C-Compiler, make sure to run their MPLAB integration tool, otherwise the PIC Compiler will not work. The MPLAB integration tool for MPLAB version 6.xx will work for MPLAB

v. 7.00. After the installation find the highlighted tool suite (under Project select Set Language Tool Locations). Make sure to use this one. Do not use CCS C Compiler or will get 'BUILD FAILED' after compiling and downloading the HEX file will not work.)

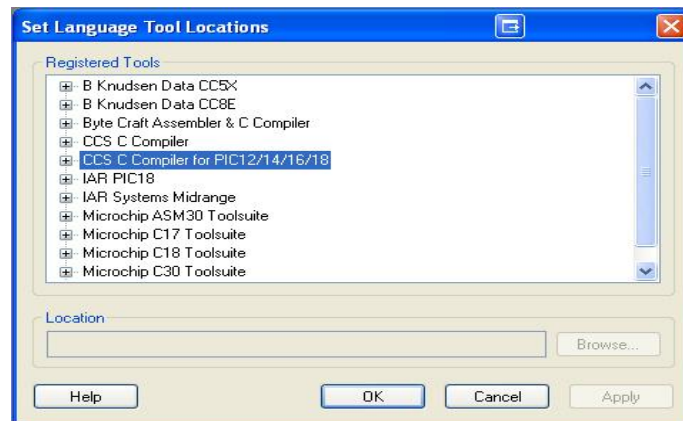


Fig A2.2 Set language tool location

Before starting own projects, make sure to configure MPLAB to not use a one-to-one-workspace setting. Then switch on this setting under Configure,Settings.

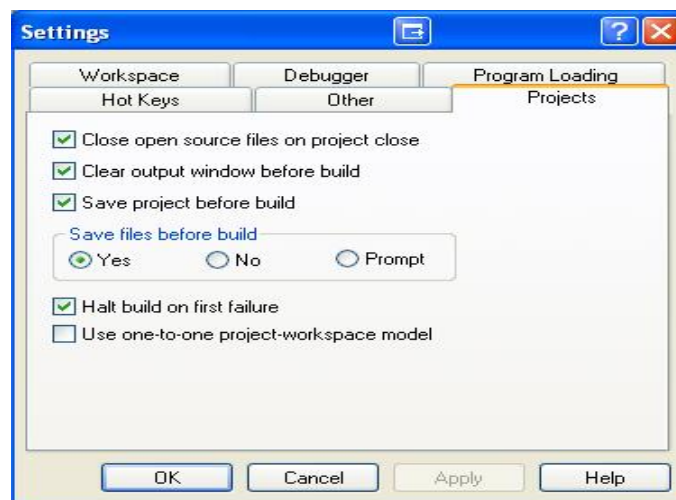


Fig A2.3 Configure Settings

Configure the configuration bits for the processor. The configuration can be done under Configure, Configuration Bits. After the workspaces and device parameters are set correctly, make a new workspace by using the Project Wizard

(Project, Project Wizard). For Smart-Its programming, the chip is the PIC18F452, for Particle programming, the chip is the PIC18F6720. Save the workspace to prevent damage to that workspace because of MPLABs stability problems.

Configure the programmer. Use a MPLAB ICD2 (ICD = in circuit programmer) programmer. Select the programmer by clicking on Programmer, Select Programmer and then choose the programmer. Go to the menu Programmer and chose Settings.

Make sure to power the Particles from the MPLAB ICD2. The programmer automatically chooses which memory ranges to write then run the self-test in the status tab and then save MPLAB workspace to make sure the programmer is selected when opening the workspace the next time.

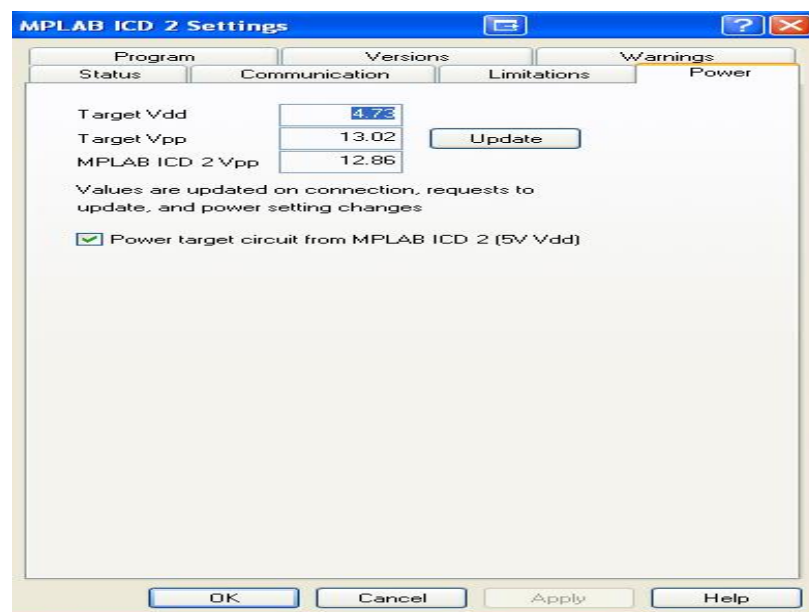


Fig A2.4 MPLAB ICD 2 settings-Power

The above figure shows how to set the power setting in MPLAB ICD. Target Vdd, Target Vpp and MPLAB ICD Vpp should be set.

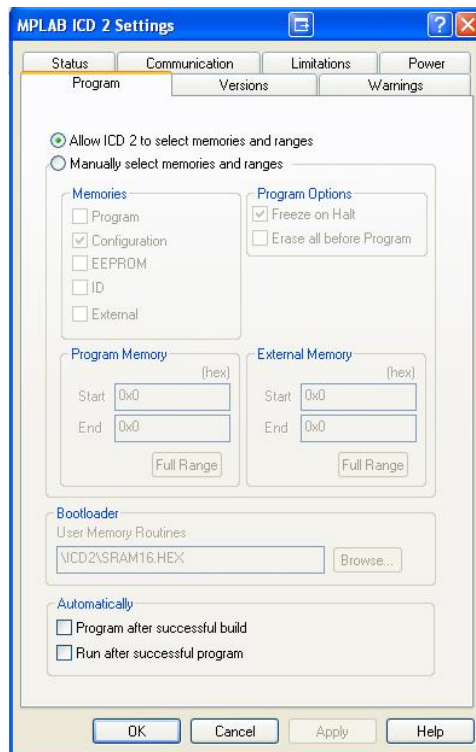


Fig A2.4 MPLAB ICD 2 settings-Program

The above figure shows the program setting of MPLAB ICD. Fig A2.4 shows the status setting property of MPLAB ICD.

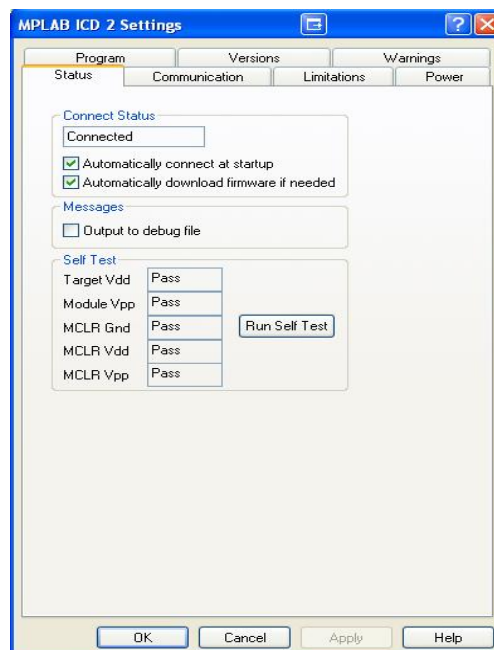


Fig A2.4 MPLAB ICD 2 settings-Status

MPLAB PROGRAMMING:

Add files to project by right-clicking on **Source Files** in Workspace (window with the title <muc.mcw>) and selecting **Add Files...**. For example, name the workspace **asmuc.mcw** and the project **muc.mcp**.

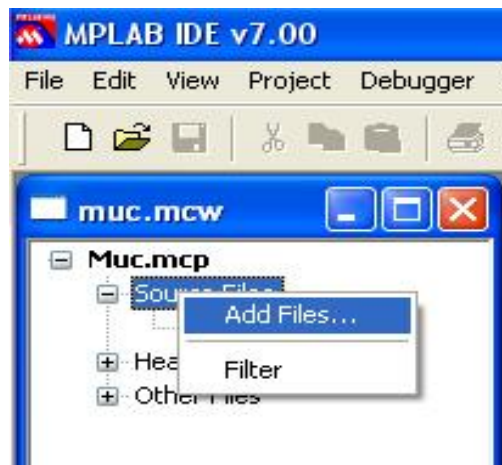


Fig A2.7 MPLAB IDE V7.00

Click right on the <muc.mcp> and select **Build Options**. Add compile settings, e.g. additional include directories.

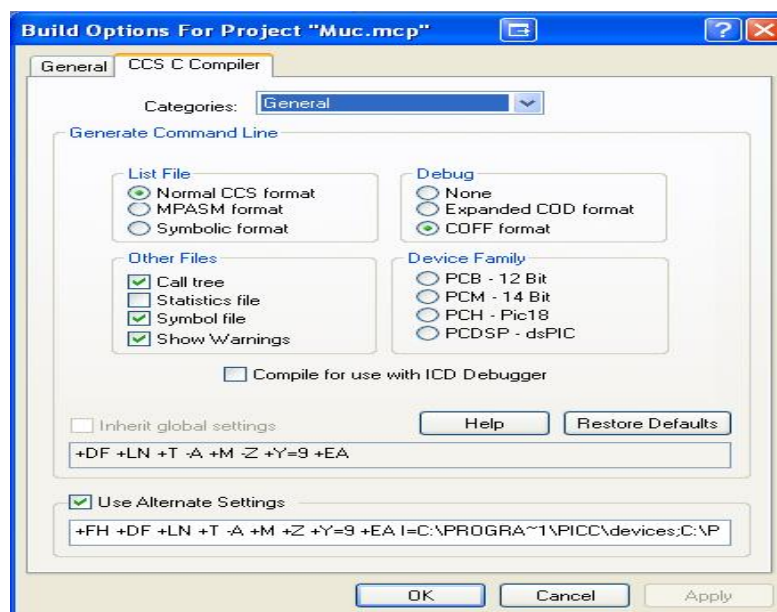


Fig A2.8 Build option for project Muc.mcp

Click on the compile symbol and compile source code. If code contains any includes (e.g. for the PIC), these files are added automatically to the section **Header Files** in project (here: muc.mcp). The output window there should read BUILD SUCCEEDED, preceded by warnings that may appear. Click on the image for a larger image.

ANDROID

Android is basically an operating system for smart phones. But to be integrated into PDAs, touch pads or televisions, even cars (trip computer) or netbooks. The OS was created by the start-up of the same name, which is owned by Google since 2005 . In this project android application is used as a user interface to interact with the system. The application is built with eclipse platform. An Android application should be installed on the user's Android mobile handset to control various home appliances. The android application accepts voice command as input.

SPECIFICATIONS

This operating system is based on version 2.6 of Linux, so it has a monolithic system kernel, what means that all system functions and drivers are grouped into one block of code.

ARCHITECTURE

Android consists of five layers:

- The Linux kernel 2.6-which includes useful drivers that allow for example WiFi or Bluetooth.
- The library written in C and C + + that provide higher level functionality such as an HTML engine, or a database (SQLite).
- A runtime environment for applications based on a virtual machine, made for

inefficient machines such as telephones. The aim is to translate JAVA in machine language understood by Android.

- A JAVA framework that allows applications running on the virtual machine to organize and cooperate.
- The user applications written in Java (Web browser, contact manager etc. ..)

ECLIPSE

Eclipse is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages via plug-ins, including Ada, ABAP, C, C++, C#, COBOL, D, Fortran, Haskell, JavaScript, Julia, Lasso, Lua, NATURAL, Perl, PHP, Prolog, Python, R, Ruby (including Ruby on Rails framework), Rust, Scala, Clojure, Groovy, Scheme, and Erlang. It can also be used to develop documents with LaTeX (via a TeXlipse plug-in) and packages for the software Mathematics. Development environments include the Eclipse Java development tools (JDT) for Java and Scala, Eclipse CDT for C/C++, and Eclipse PDT for PHP, among others.

Eclipse software development kit (SDK) is free and open-source software, released under the terms of the Eclipse Public License, although it is incompatible with the GNU General Public License. It was one of the first IDEs to run under GNU Class path and it runs without problems under IcedTea.

INSTALL THE JAVA RUN-TIME ENVIRONMENT

Java Run-Time Environment must be installed in order to run a java application. Through Oracle website to JRE can be downloaded. Choose to download either a JRE or a JDK. For most of the functions in Eclipse, a JRE should be sufficient. But to develop a Dynamic Web Project, and to debug it with a J2EE Preview server, JDK is needed to power the Eclipse IDE. Choose the JDK version (Java SE 8U5) for windows 64-bit computers and installed it to the default location.

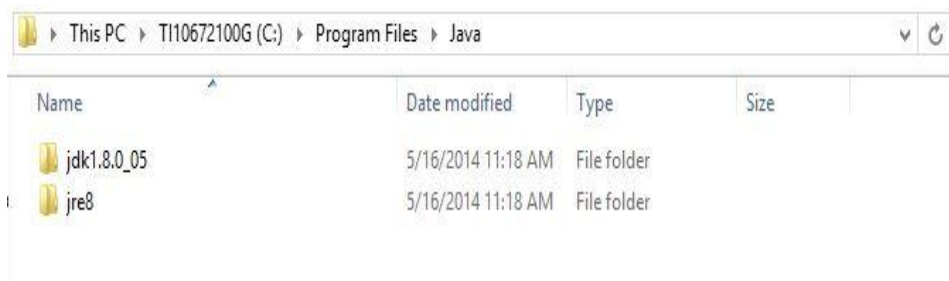


Fig A2.9 JDK Installation

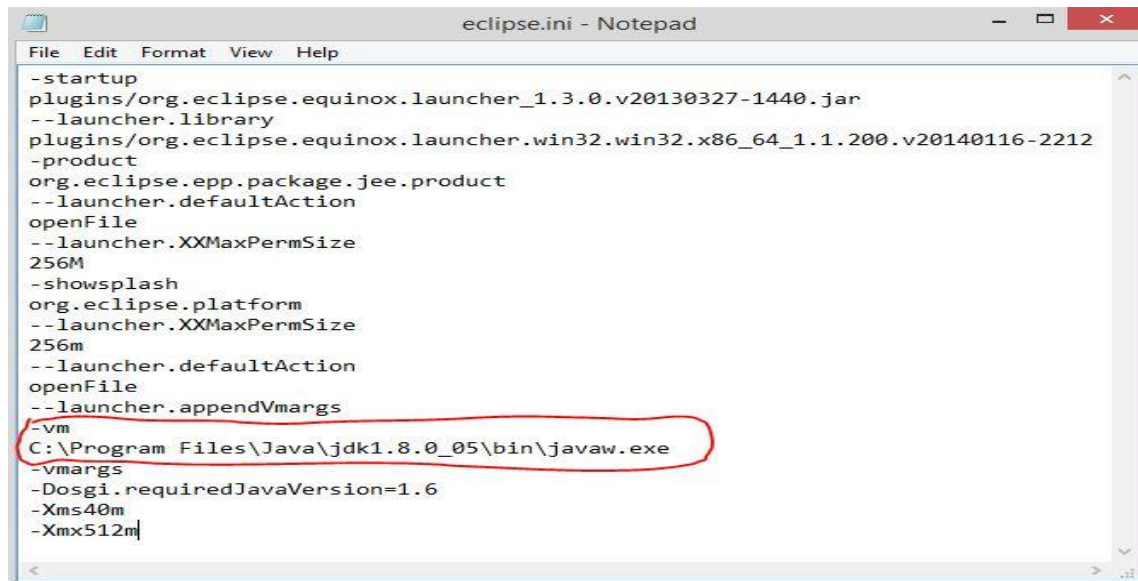
The installer added both JRE and JDK to the computer. This location need to be remembered and later configure Eclipse to use this particular version of Java Run-time environment. Choose the Desired Java Run-time to Launch Eclipse.

It is very common that there are multiple versions of Java Run-time environment on the computer. It is desirable to specify the desired one to power Eclipse. According to the Eclipse documentation, there are three common methods to specify the Java Run-time environment.

- If a Java Run-time environment is installed in the eclipse/jre directory, Eclipse will use it;
- Otherwise the Eclipse launcher will consult the "eclipse.ini" file and the system path variable. Eclipse does not consult the JAVA_HOME environment variable;

- Choose to directly invoke the desired JVM to start the "jar" file that launches Eclipse. For the version of Eclipse, it is the "org.eclipse.equinox.launcher_1.3.0.v20130327-1440.jar" file in the "plugins" folder.

According to the Eclipse documentation, the most recommended method is to specify the desired Java Run-time in the "eclipse.ini" file.



```
eclipse.ini - Notepad
File Edit Format View Help
-startup
plugins/org.eclipse.equinox.launcher_1.3.0.v20130327-1440.jar
--launcher.library
plugins/org.eclipse.equinox.launcher.win32.win32.x86_64_1.1.200.v20140116-2212
-product
org.eclipse.epp.package.jee.product
--launcher.defaultAction
openFile
--launcher.XXMaxPermSize
256M
-showsplash
org.eclipse.platform
--launcher.XXMaxPermSize
256m
--launcher.defaultAction
openFile
--launcher.appendVmargs
-vm
C:\Program Files\Java\jdk1.8.0_05\bin\javaw.exe
-vmargs
-Dosgi.requiredJavaVersion=1.6
-Xms40m
-Xmx512m
```

FIG A2.10 Path setup

In study note, the "-vm" parameter in the "eclipse.ini" file pointing to the JVM that are just installed. After saving the "eclipse.ini" file, it is ready to be launched.

Launch Eclipse

Eclipse organizes the code in "workspaces". A "workspace" is a folder where Eclipse put all your code for your projects. For example, to launch Eclipse, let us create an empty folder called "FirstEclipseWorkspace" anywhere in the computer.

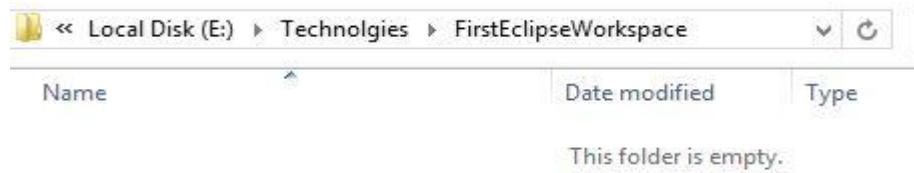


Fig A2.11 Workspace

In the windows environment, double click the "eclipse.exe" file to launch Eclipse.

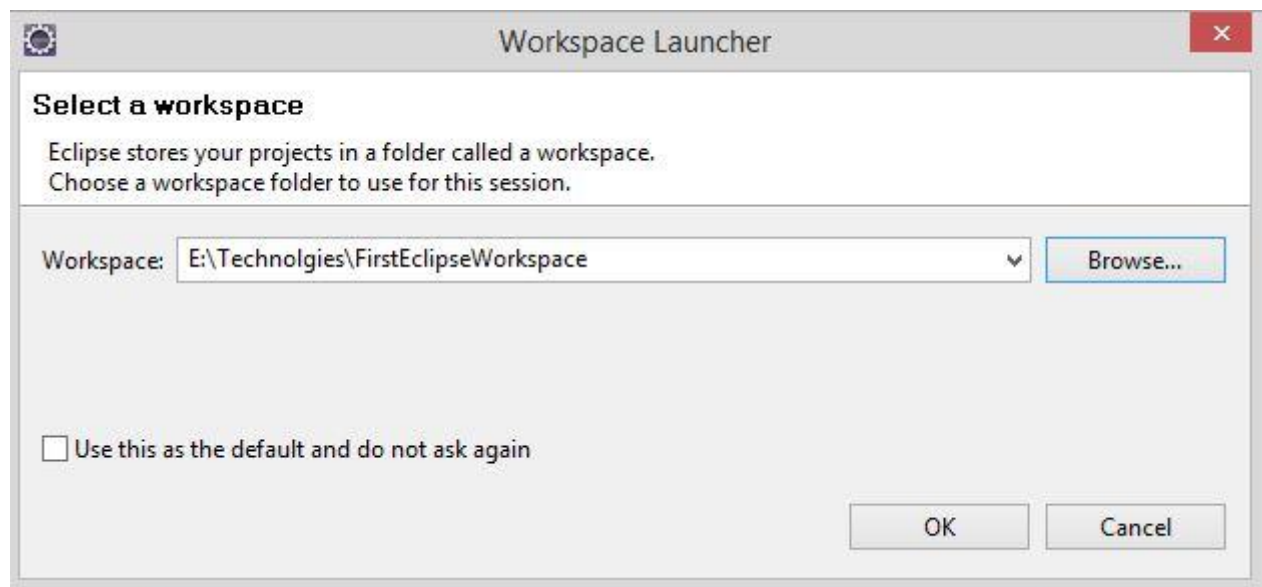


Fig A2.12 Workspace Launcher

During the launching process, Eclipse will ask to select a "workspace". Browse to the folder just created and click the "OK" button.



Fig A2.13 Eclipse IDE

Since the "FirstEclipseWorkspace" folder is empty, Eclipse shows the welcome page. To double check that Eclipse is indeed powered by the desired version of JVM, click on the "Help" menu item and then click on "About Eclipse" to open the "About Eclipse" window.



Fig A2.14 About Eclipse

In the "About Eclipse" window, clicking on the "Installation Details" button will open the "Eclipse Installation Details" window.

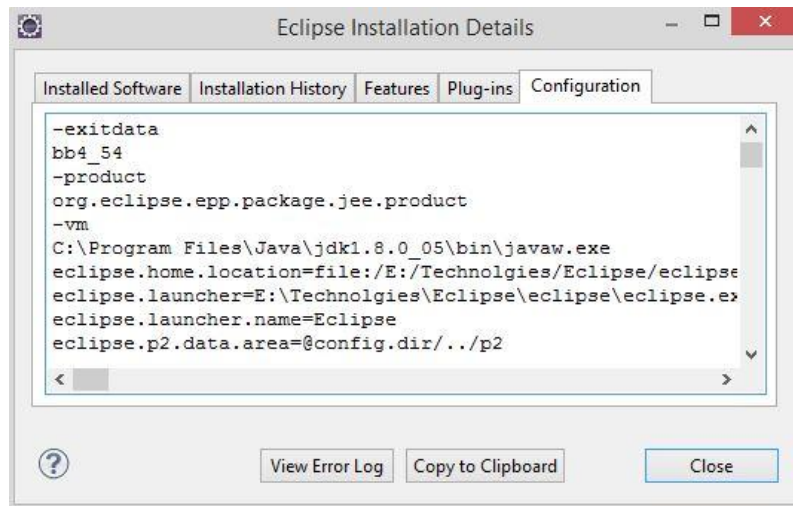


Fig A2.15 Eclipse Installation Details

See that the JVM that powers this instance of Eclipse is indeed the JVM that is specified in the "eclipse.ini" file.

APPENDIX 3

SOURCE CODE

ANDROID CODE

BLURTOOTH CONNECTIVITY

```
public static final String TOAST = "toast";

// Intent request codes

private static final int REQUEST_CONNECT_DEVICE = 1;

private static final int REQUEST_ENABLE_BT = 2;

// Layout Views

private TextView mTitle;

private ListView mConversationView;

    private EditText mOutEditText;

private Button mSendButton;

    // Name of the connected device

    private String mConnectedDeviceName = null;

// Array adapter for the conversation thread

private ArrayAdapter<String> mConversationArrayAdapter;

    // String buffer for outgoing messages

private StringBuffer mOutStringBuffer;

    // Local Bluetooth adapter

private BluetoothAdapter mBluetoothAdapter = null;

    // Member object for the chat services
```

```

private BluetoothChatService mChatService = null;

@Override

public void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);

    if(D) Log.e(TAG, "+++ ON CREATE +++");

    // Set up the window layout

    requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);

    setContentView(R.layout.main);

    getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE,
R.layout.custom_title);

    // Set up the custom title

    mTitle = (TextView) findViewById(R.id.title_left_text);

    mTitle.setText(R.string.app_name);

    mTitle = (TextView) findViewById(R.id.title_right_text);

    // Get local Bluetooth adapter

    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    // If the adapter is null, then Bluetooth is not supported

    if (mBluetoothAdapter == null) {

        Toast.makeText(this, "Bluetooth is not available",
Toast.LENGTH_LONG).show();} }

```

BLUETOOTH CONNECTIVITY:

```

import java.io.IOException;

import java.io.InputStream;

import java.io.OutputStream;

import java.util.UUID;

```

```

import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothServerSocket;
import android.bluetooth.BluetoothSocket;
import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
/**
 * This class does all the work for setting up and managing Bluetooth
 * connections with other devices. It has a thread that listens for
 * incoming connections, a thread for connecting with a device, and a
 * thread for performing data transmissions when connected.
 */
public class BluetoothChatService {
    // Debugging
    private static final String TAG = "BluetoothChatService";
    private static final boolean D = true;
    // Name for the SDP record when creating server socket
    private static final String NAME = "BluetoothChat";
    // Unique UUID for this application
    private static final UUID MY_UUID = UUID.fromString("fa87c0d0-afac-11de-8a39-0800200c9a66");
    // Member fields
    private final BluetoothAdapter mAdapter;
    private final Handler mHandler;

```

```

private AcceptThread mAcceptThread;
private ConnectThread mConnectThread;
private ConnectedThread mConnectedThread;
private int mState;

// Constants that indicate the current connection state

public static final int STATE_NONE = 0;    // we're doing nothing
public static final int STATE_LISTEN = 1;    // now listening for incoming
connections
public static final int STATE_CONNECTING = 2; // now initiating an outgoing
connection
public static final int STATE_CONNECTED = 3; // now connected to a remote
device

/**
 * Constructor. Prepares a new BluetoothChat session.
 * @param context The UI Activity Context
 * @param handler A Handler to send messages back to the UI Activity
 */

public BluetoothChatService(Context context, Handler handler) {
mAdapter = BluetoothAdapter.getDefaultAdapter();
mState = STATE_NONE;
mHandler = handler;
} /**
 * Set the current state of the chat connection
 * @param state An integer defining the current connection state
 */

private synchronized void setState(int state) {
if (D) Log.d(TAG, "setState() " + mState + " -> " + state);

```



```

mState = state;

// Give the new state to the Handler so the UI Activity can update
    mHandler.obtainMessage(BluetoothChat.MESSAGE_STATE_CHANGE, state,
-1).sendToTarget();

} /**

    * Return the current connection state. */

public synchronized int getState() {
return mState;

}

/**

    * Start the chat service. Specifically start AcceptThread to begin a
    * session in listening (server) mode. Called by the Activity onResume() */

public synchronized void start() {
    if (D) Log.d(TAG, "start");

// Cancel any thread attempting to make a connection
    if (mConnectThread != null)
    {mConnectThread.cancel();

mConnectThread = null;}

    // Cancel any thread currently running a connection

    if (mConnectedThread != null)
    {mConnectedThread.cancel();

mConnectedThread = null;}

    // Start the thread to listen on a BluetoothServerSocket

```

```

    if (mAcceptThread == null) {
        mAcceptThread = new AcceptThread();
        mAcceptThread.start();
    }
    setState(STATE_LISTEN);
} /**
 * Start the ConnectThread to initiate a connection to a remote device.
 * @param device The BluetoothDevice to connect
 */

public synchronized void connect(BluetoothDevice device) {
    if (D) Log.d(TAG, "connect to: " + device);
    // Cancel any thread attempting to make a connection
    if (mState == STATE_CONNECTING) {
        if (mConnectThread != null)
        {mConnectThread.cancel();
        mConnectThread = null;} }
    // Cancel any thread currently running a connection
    if (mConnectedThread != null)
    {mConnectedThread.cancel();
    mConnectedThread = null;}

    // Start the thread to connect with the given device
    mConnectThread = new ConnectThread(device);
    mConnectThread.start();
    setState(STATE_CONNECTING);
} /**
 * Start the ConnectedThread to begin managing a Bluetooth connection

```

```

    * @param socket The BluetoothSocket on which the connection was mad *
    @param device The BluetoothDevice that has been connected

    */

    public synchronized void connected(BluetoothSocket socket, BluetoothDevice
device) {

        if (D) Log.d(TAG, "connected");
        // Cancel the thread that completed the connection
        if (mConnectThread != null)
        {mConnectThread.cancel();
        mConnectThread = null;}

        // Cancel any thread currently running a connection
        if (mConnectedThread != null)
        {mConnectedThread.cancel();
        mConnectedThread = null;}

        // Cancel the accept thread because we only want to connect to one device
        if (mAcceptThread != null)
        {mAcceptThread.cancel();
        mAcceptThread = null;}

        // Start the thread to manage the connection and perform transmissions
        mConnectedThread = new ConnectedThread(socket);
        mConnectedThread.start();

        // Send the name of the connected device back to the UI Activity
        Message
msg = mHandler.obtainMessage(BluetoothChat.MESSAGE_DEVICE_NAME);
        Bundle bundle = new Bundle();
        bundle.putString(BluetoothChat.DEVICE_NAME, device.getName());

```

```

msg.setData(bundle);
mHandler.sendMessage(msg);
setState(STATE_CONNECTED); }

/**
 * Stop all threads
 */

public synchronized void stop() {
if (D) Log.d(TAG, "stop");
if (mConnectThread != null)
{ mConnectThread.cancel();
mConnectThread = null;}
if (mConnectedThread != null)
{ mConnectedThread.cancel();
mConnectedThread = null;}
if (mAcceptThread != null)
{ mAcceptThread.cancel();
mAcceptThread = null;}
setState(STATE_NONE);
} /**

 * Write to the ConnectedThread in an unsynchronized manner
 * @param out The bytes to write
 * @see ConnectedThread#write(byte[])
 */

public void write(byte[] out) {
// Create temporary object
ConnectedThread r;

// Synchronize a copy of the ConnectedThrea

```

```

synchronized (this) {
    if (mState != STATE_CONNECTED) return;
    r = mConnectedThread;
}

// Perform the write unsynchronized
r.write(out); }

/**
 * Indicate that the connection attempt failed and notify the UI Activity.
 */

private void connectionFailed() {
    setState(STATE_LISTEN);
    // Send a failure message back to the Activity
    Message msg = mHandler.obtainMessage(BluetoothChat.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString(BluetoothChat.TOAST, "Unable to connect device");
    msg.setData(bundle);
    mHandler.sendMessage(msg);
} /**
 * Indicate that the connection was lost and notify the UI Activity.
 */

private void connectionLost() {
    setState(STATE_LISTEN);
    // Send a failure message back to the Activity
    Message msg = mHandler.obtainMessage(BluetoothChat.MESSAGE_TOAST);
    Bundle bundle = new Bundle();
    bundle.putString(BluetoothChat.TOAST, "Device connection was lost");
    msg.setData(bundle);

```

```

mHandler.sendMessage(msg); }

/**
 * This thread runs while listening for incoming connections. It behaves
 * like a server-side client. It runs until a connection is accepted
 * (or until cancelled).
 */

private class AcceptThread extends Thread {
    // The local server socket

    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        BluetoothServerSocket tmp = null;

        // Create a new listening server socket

        try {s
            tmp = mAdapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID); }
        catch (IOException e) {
            Log.e(TAG, "listen() failed", e);
        }

        mmServerSocket = tmp;
    }

    public void run() {
        if (D) Log.d(TAG, "BEGIN mAcceptThread" + this);
        setName("AcceptThread");

        BluetoothSocket socket = null;

        // Listen to the server socket if we're not connected

        while (mState != STATE_CONNECTED) {
            try {

                // This is a blocking call and will only return on a
                // successful connection or an exception

```

```

socket = mmServerSocket.accept();
    } catch (IOException e) {
        Log.e(TAG, "accept() failed", e);
        break;
    }
// If a connection was accepted
if (socket != null) {
    synchronized (BluetoothChatService.this) {
        switch (mState) {
            case STATE_LISTEN:
            case STATE_CONNECTING:
                // Situation normal. Start the connected thread.
                connected(socket, socket.getRemoteDevice());
            break;
            case STATE_NONE:
            case STATE_CONNECTED:
                // Either not ready or already connected. Terminate new socket.
                try {
                    socket.close();
                } catch (IOException e) {
                    Log.e(TAG, "Could not close unwanted socket", e);
                }
            break; }
}

```

APPENDIX 4

SCREENSHOTS

HARDWARE SETUP

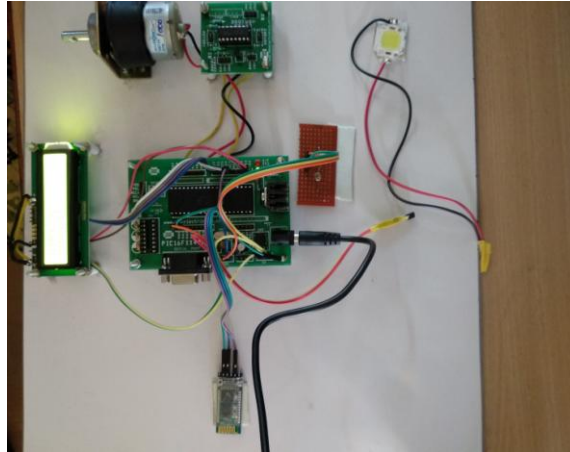


Fig A4.1 Hardware Setup

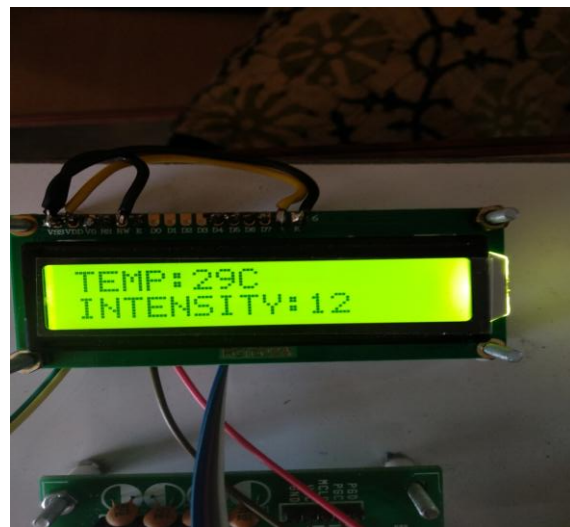
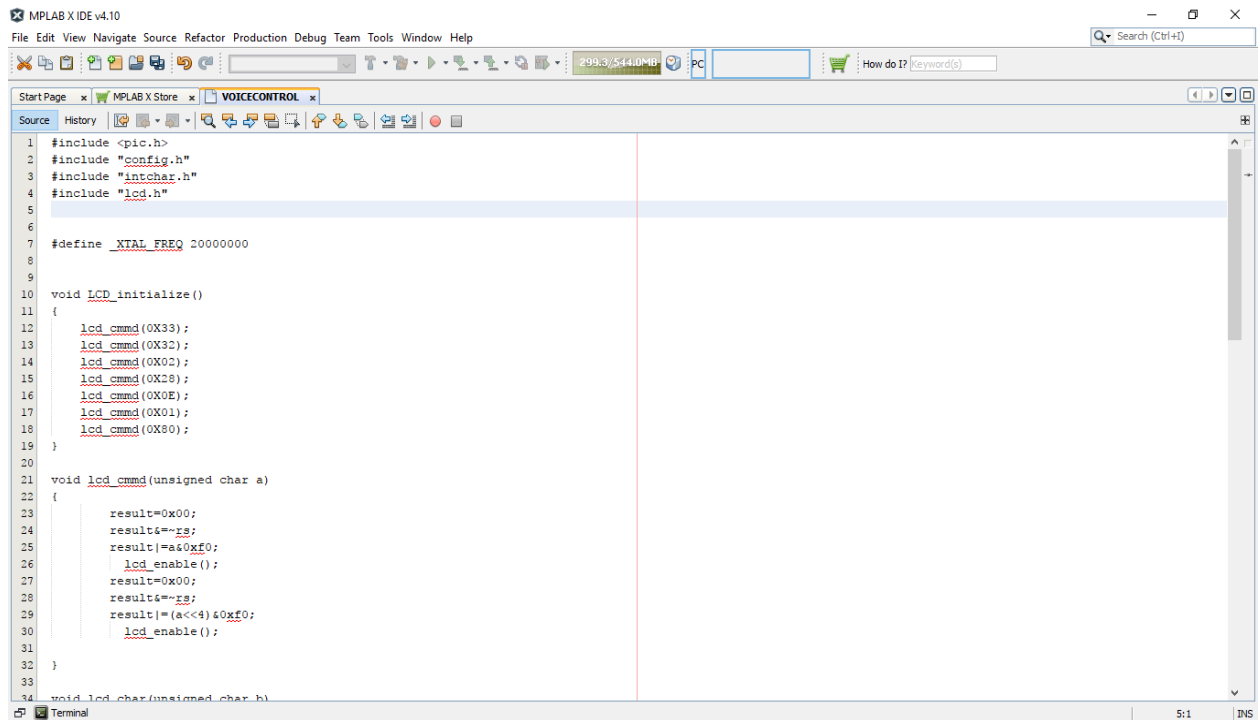


Fig A4.2 Sensor Readings

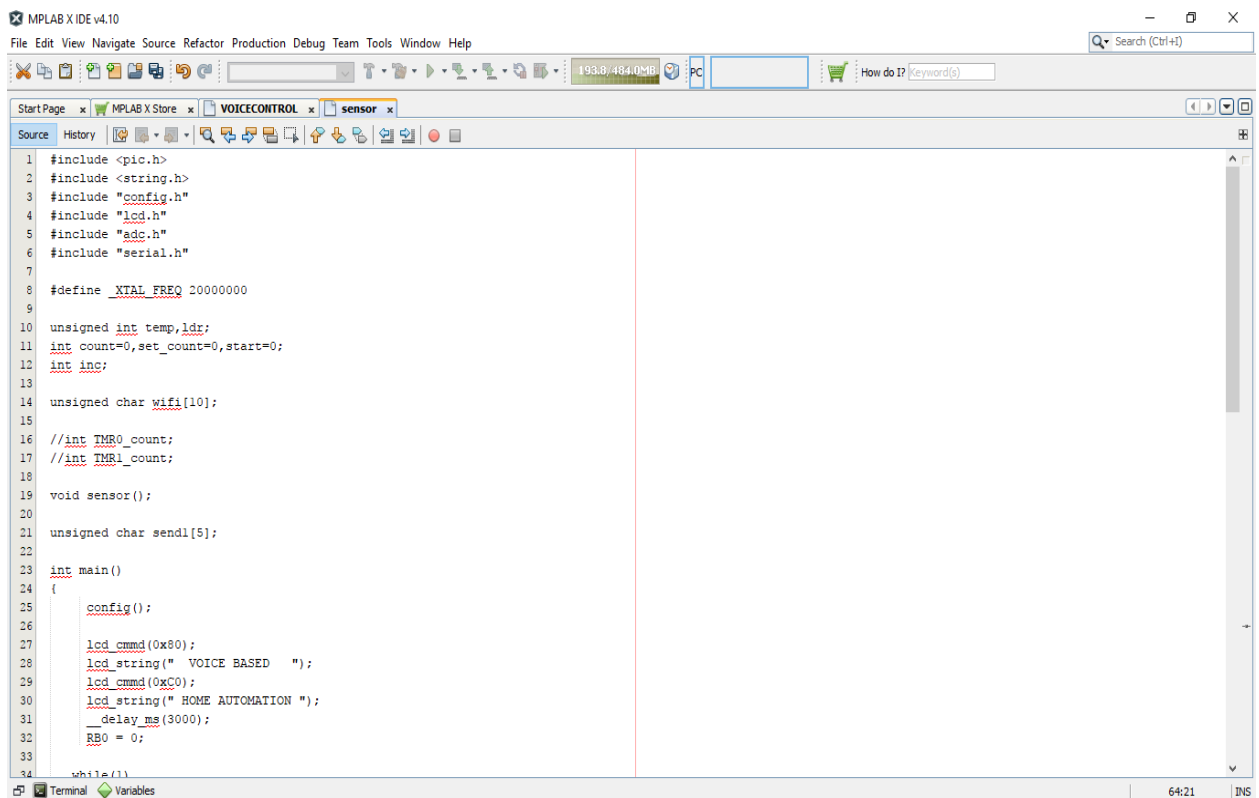
CODING



The screenshot shows the MPLAB X IDE v4.10 interface. The main window displays the source code for a project named "VOICECONTROL". The code includes headers for PIC, config, intchar, and lcd. It defines a macro for XTAL_FREQ as 20000000. The function void LCD_initialize() calls lcd_cmd with various commands. The function void lcd_cmd(unsigned char a) handles the command byte, setting the result and enabling the LCD. The function void lcd_char(unsigned char b) is also shown.

```
1 #include <pic.h>
2 #include "config.h"
3 #include "intchar.h"
4 #include "lcd.h"
5
6
7 #define XTAL_FREQ 20000000
8
9
10 void LCD_initialize()
11 {
12     lcd_cmd(0X33);
13     lcd_cmd(0X32);
14     lcd_cmd(0X02);
15     lcd_cmd(0X28);
16     lcd_cmd(0X0E);
17     lcd_cmd(0X01);
18     lcd_cmd(0X80);
19 }
20
21 void lcd_cmd(unsigned char a)
22 {
23     result=0x00;
24     result|=~rg;
25     result|=a&0xf0;
26     lcd_enable();
27     result=0x00;
28     result|=~rg;
29     result|=(a<<4)&0xf0;
30     lcd_enable();
31 }
32
33
34 void lcd_char(unsigned char b)
```

Fig A4.3 MPLABX Code for LCD



The screenshot shows the MPLAB X IDE v4.10 interface. The main window displays the source code for a project named "VOICECONTROL". The code includes headers for PIC, string, config, lcd, adc, and serial. It defines a macro for XTAL_FREQ as 20000000. The code declares variables for temp, ldr, count, set_count, start, inc, wifi, TMR0_count, TMR1_count, sendl, and main. The function void sensor() is shown. The main function calls config, lcd_cmd, lcd_string, and delay_ms.

```
1 #include <pic.h>
2 #include <string.h>
3 #include "config.h"
4 #include "lcd.h"
5 #include "adc.h"
6 #include "serial.h"
7
8 #define XTAL_FREQ 20000000
9
10 unsigned int temp,ldr;
11 int count=0,set_count=0,start=0;
12 int inc;
13
14 unsigned char wifi[10];
15
16 //int TMR0_count;
17 //int TMR1_count;
18
19 void sensor();
20
21 unsigned char sendl[5];
22
23 int main()
24 {
25     config();
26     lcd_cmd(0x80);
27     lcd_string(" VOICE BASED ");
28     lcd_cmd(0xc0);
29     lcd_string(" HOME AUTOMATION ");
30     delay_ms(3000);
31     RB0 = 0;
32
33     while(1)
```

Fig A4.4 MPLABX Code for Sensor

REFERENCES

1. M. R. Alam, M. B. I. Reaz, and M. A. M. Ali (2012), 'A Review of Smart Homes— Past, Present, and Future', IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), Vol. 42(6): pp.11901203.
2. Nagender Kumar Suryadevara, Subhas Chandra Mukhopadhyay, Sean Dieter Tebje Kelly, and Satinder Pal Singh Gill, (2015), 'WSN-Based Smart Sensors and Actuator for Power Management in Intelligent Buildings', IEEE/ASME TRANSACTIONS ON MECHATRONICS, VOL. 20, NO. 2.
3. Lin Gao¹, Zhixin Wang¹, Jianlong Zhou², Chao Zhang³, (2016), 'Design of Smart Home System Based on ZigBee Technology and R&D for Application', Published Online in SciRes.
4. Murad Khan, BhagyaNathali Silva, Kijun Han, 'Internet of Things based Energy Aware Smart Home Control System', Citation information: DOI 10.1109/ACCESS.2016.2621752, IEEEAccess.
5. Prateek Gupta, 'Human Voice Controlled Home Appliances', Student, Electronics and Telecommunication Engineering, Bharati Vidyapeeth University C.O.E, Pune, India
6. NorhafizahbtAripin and M. B. Othman, (2014), 'Voice Control of Home Appliances using Android' ,Electrical Power, Electronics, Communications, Controls, and Informatics Seminar (EECCIS) 978.
7. Syed Ali Imran Quadri, P.Sathish , 'IoT Based Home Automation and Surveillance System', Department of Electronics and Communication Engineering, ChaitanyaBharathi Institute of Technology, Gandipet, Hyderabad,India

8. N. Sriskanthan*, F. Tan, A. Karande, 'Bluetooth based home automation system', School of Computer Engineering, Nanyang Technological University, Nanyangavenue, Singapore, Singapore.
9. Ming Yan and Hao, 'Smart Living Using Bluetooth Based Android Smartphone', Shi College of Engineering and Science, VictoriaUniversity, Melbourne,Australia.