HOME AUTOMATION SYSTEM BASED ON BLUETOOTH, GSM AND ETHERNET

Dissertation submitted in fulfillment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

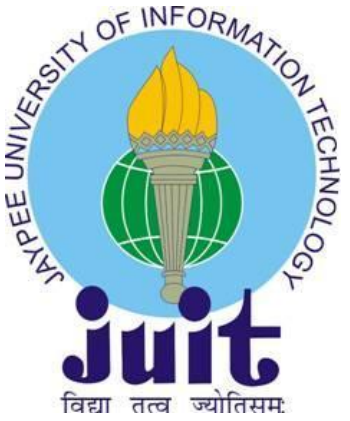
IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By

AVIRAL DIXIT
AYANTI SENGUPTA
TARANDEEP KALRA

UNDER THE GUIDANCE OF
Prof. T. S. LAMBA



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT
May, 2016

HOME AUTOMATION SYSTEM BASED ON BLUETOOTH, GSM AND ETHERNET

Dissertation submitted in fulfillment of the requirements for the Degree of

BACHELOR OF TECHNOLOGY

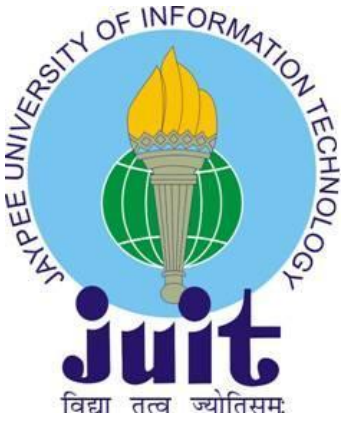
IN

ELECTRONICS AND COMMUNICATION ENGINEERING

By

AVIRAL DIXIT
AYANTI SENGUPTA
TARANDEEP KALRA

UNDER THE GUIDANCE OF
Prof. T. S. LAMBA



JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT
May, 2016

TABLE OF CONTENTS

	Page Number
ABSTRACT	**vi**
ACKNOWLEDGEMENT	**vii**
DECLARATION BY THE SCHOLAR	**viii**
SUPERVISOR'S CERTIFICATE	**ix**
LIST OF ACRONYMS AND ABBREVIATIONS	**x**
LIST OF FIGURES	**xi**
CHAPTER-1	
HOME AUTOMATION SYSTEM	**1**
1. 1 WHAT IS A HOME AUTOMATION SYSTEM?	**1**
1. 2 SYSTEM LAYOUT	**2**
CHAPTER-2	
THE ARDUINO UNO	**3**
2. 1 HARDWARE	**3**
2.1.1 ATMEGA328P MICROCONTROLLER	**3**
2.1.2 DIGITAL PINS	**4**
2.1.3 ANALOG PINS	**4**
2.1.4 POWER PINS	**5**
2.2 ARDUINO SOFTWARE (IDE)	**5**
2.2.1 FEATURES	**6**

CHAPTER-3

BLUETOOTH **7**

3. 1 THE BLUETOOTH TECHNOLOGY **7**

3.2 BLUETOOTH MODULE **8**

3.3 MODES OF OPERATION **9**

3.4 AT COMMANDS FOR BT-MODULE **10**

3.5 INTERFACING BT-MODULE AND ARDUINO **11**

CHAPTER-4

GSM **12**

4.1 GSM (SIM900) BOARD **13**

4.2 FEATURES OF THE BOARD **15**

4.3 AT COMMANDS FOR GSM MODULE **16**

4.4 INTERFACING THE GSM MODULE AND ARDUINO **17**

CHAPTER-5

ETHERNET **18**

5.1 THE ETHERNET SHIELD **18**

CHAPTER-6

THE ANDROID APP **21**

6.1 THE HAS APP PAGE 1: BLUETOOTH **21**

6.2 THE HAS APP PAGE 2: GSM **22**

6.3 THE HAS APP PAGE 3: ETHERNET **23**

CHAPTER-7

RELAYS **25**

7.1 DEFINING RELAYS **25**

7.2 WORKING OF RELAYS **26**

CHAPTER-8

CODE **27**

8.1 CODE FOR ARDUINO UNO BOARD **27**

8.2 CODE FOR HAS APP: BLUETOOTH **34**

8.3 CODE FOR HAS APP: GSM **35**

8.4 CODE FOR HAS APP: ETHERNET **36**

CONCLUSION **37**

REFERENCES **38**

ABSTRACT

With the rising power of technology, we are able to accomplish tasks with increased comfort and at a much quicker rate. Technology has provided us with the ability to communicate, organize, and manage our time in a more efficient manner. Automation of the surrounding environment of a modern human being allows increasing the work efficiency and comfort. The "**Home Automation**" concept has existed for many years. The terms "Smart Home" and "Intelligent Home" have been used to introduce the concept of networking appliances and devices in the house. The aim of this project is to enable communication between the user and different electrical devices within the home wirelessly. This project is a combined utilization of an Arduino Uno board with the microcontroller(Atmega 328P-PU), a Bluetooth module, a GSM module(SIM 900A), an Ethernet Shield and an android device to create a wireless network that connects the home appliances as a single unit for controlling household appliances remotely using a smartphone. This project is particularly useful for physically disabled/old age people who have difficulty moving and would appreciate the ability to switch ON/OFF their home appliances with minimal movement. It can also be useful in situations where forgetting to turn OFF any home appliance on leaving the home may be corrected and will also enable the user to check the status of the appliances and control them. It may be useful in winters/summers to pre warm/cool the home, if required.

The android app for the HAS has been developed using the online software MIT app inventor 2. With the help of the android app we can control and monitor any appliance through arduino, and thus establish a successful communication link between the mobile phone and the microcontroller. Home Automation is undeniably a resource which can make our lives easier with increased comfort and efficiency. The nature of this project is such that it provides a great scope for further developments.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our advisor Prof. T. S. Lamba. We have been fortunate to have an advisor who gave us the freedom to explore the all the possibilities of this project on our own and at the same time offered guidance to recover when our steps faltered. He has supported us throughout our thesis with his patience, motivation and knowledge.

We are grateful to Mr. Manoj Kumar Pandey (ECE lab1), Mr. Pramod Kumar (ECE lab 5) and Mr. Mohan Sharma (Project Lab) who aided in the use of equipment and also provided valuable suggestions during the initial phase of our thesis work.

We would also like to take this opportunity to express gratitude to all of the department faculty members in the 4years of our undergraduate study for it is their support and encouragement that motivated us to work on this project.

DECLARATION BY THE SCHOLAR

I hereby declare that the work reported in the B-Tech thesis entitled **"Home Automation System based on Bluetooth, GSM and Ethernet"** submitted at **Jaypee University of Information Technology, Waknaghat, India,** is an authentic record of my work carried out under the supervision of **Prof. T.S. Lamba**. I have not submitted this work elsewhere for any other degree or diploma.

Aviral Dixit

Ayanti Sengupta

Tarandeep Kalra

Department of Electronics and Communication Engineering

Jaypee University of Information Technology, Waknaghat, India

Date: 26th May, 2016

SUPERVISOR'S CERTIFICATE

This is to certify that the work reported in the B-Tech. thesis entitled "**Home Automation System based on Bluetooth, GSM and Ethernet** at **Jaypee University of Information Technology, Waknaghat**, **India,** is a bonafide record of his / her original work carried out under my supervision. This work has not been submitted elsewhere for any other degree or diploma.

Prof. T. S. Lamba

Dean (Academic and Research)

Date: 26th May, 2016

LIST OF ACRONYMS & ABBREVIATIONS

AVR	Advanced Virtual RISC
EEPROM	Electrically Erasable Programmable Read-Only Memory
FTDI	Future Technology Device International
GSM	Global System for Mobile communication
HAS	Home Automation System
ICSP	In-Circuit Serial Programming
IDE	Integrated Development Environment
ISP	Internet Service Provider
MIPS	Million Instructions Per Second
MIT	Massachusetts Institute of Technology
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
TTL	Transmitter-Transmitter Logic
URL	Uniform Resource Locator
USART	Universal Synchronous/Asynchronous Receiver/Transmitter
USB	Universal Serial Bus

LIST OF FIGURES

Figure Number	Caption	Page Number
1.1	Home Automation System Layout	2
2.1	The Arduino Uno Pinout	3
3.1	Bluetooth Module	9
3.2	Interfacing BT-module and Arduino	12
4.1	Pinout of GSM Module	14
4.2	Interfacing GSM module and Arduino	17
5.1	Ethernet Shield	19
6.1	HAS APP: Bluetooth Screen 1	25
6.2	HAS APP: GSM Screen 2	26
6.3	HAS APP: Ethernet Screen 3	31
7.1	Working of Relays	33

CHAPTER 1: HOME AUTOMATION SYSTEM

INTRODUCTION

Imagine how helpful it will be to switch on the air conditioning system ten minutes before entering your home on a hot afternoon in the month of June. What if you realize that you have left the lights on after leaving your home? How much savings will be made if you could turn it off immediately than having to wait till you get back home?

The Home Automation System (HAS) allows its users to control home appliances such as lights, fans, AC's etc., wirelessly, using an android smartphone. In this project we have used three major components which include the bluetooth module (HC-05), the GSM module (SIM 900) and the ethernet shield.

While bluetooth is the preferred communication link within the home, the GSM module will allow the user to control the appliances outside the home using the GSM mobile network. Home automation becomes truly "smart" when it is internet-enabled and it becomes possible to attach devices to this network and control it. The ethernet shield in this project serves this purpose.

1.1 WHAT IS HOME AUTOMATION SYSTEM?

A Home automation system is composed of hardware, communication and electronic interfaces that work to integrate electrical devices with one another. Domestic activities such as switching ON/OFF lights, fans etc., can then be regulated with the touch of a button.

There are several benefits of a home automation system. Some of them include:

- **Savings:** Ability to turn OFF connected devices such as A.C's, heaters and other power consuming devices remotely lead to significant energy savings.
- **Control:** Many devices inside the home, from ovens and fridges to deadbolts and garage doors, can be controlled remotely via Smartphone's and Tablets. This control also works out of the home, which means it allows the user to close the garage door from the airport or confirm that the geyser is switched off from the grocery store.

- **Convenience:** Turning ON the fans without having to reach out to the switchboard, cooling the bedroom before arriving home are certainly convenient and are also useful to the elderly or physically disabled.

1.2 SYSTEM LAYOUT

The HAS consists of the Ethernet shield board interfaced above the arduino uno board which is connected to the Bluetooth module, GSM module and the relay module. The system is based on serial data transmission using the android Smartphone. A command from individual modules is sent to the arduino uno i.e a stream of characters for the Bluetooth module, a SMS for the GSM module and the URL for the Ethernet shield and the corresponding pin is made HIGH which switches ON the relay corresponding to the appliance and the appliance is turned ON. A user interface (UI) on the Android enabled mobile phone offers system connection and control utilities.

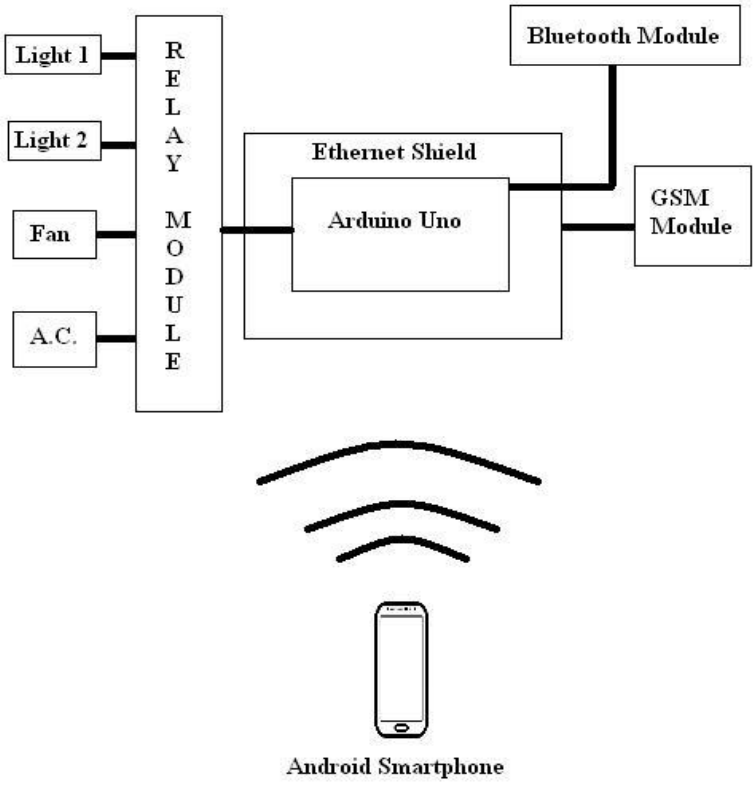


Figure 1.1: The Home Automation System Layout

CHAPTER 2: THE ARDUINO UNO

INTRODUCTION

The HAS has been built using several components, most important being the Arduino Uno board for it is a microcontroller board based on the ATmega328P. "Uno" means one in Italian and was chosen to mark the release of Arduino Software (IDE) 1.0. The code for this project has been developed using the Arduino IDE in the embedded C language. This board is the brain of the system as it is able to make decisions based on the values received by the various other components of the system.

2.1 Hardware

The Arduino Uno board has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller. The board can be connected to a computer with a USB cable or power may be provided with a AC-to-DC adapter or a battery to get the board started.

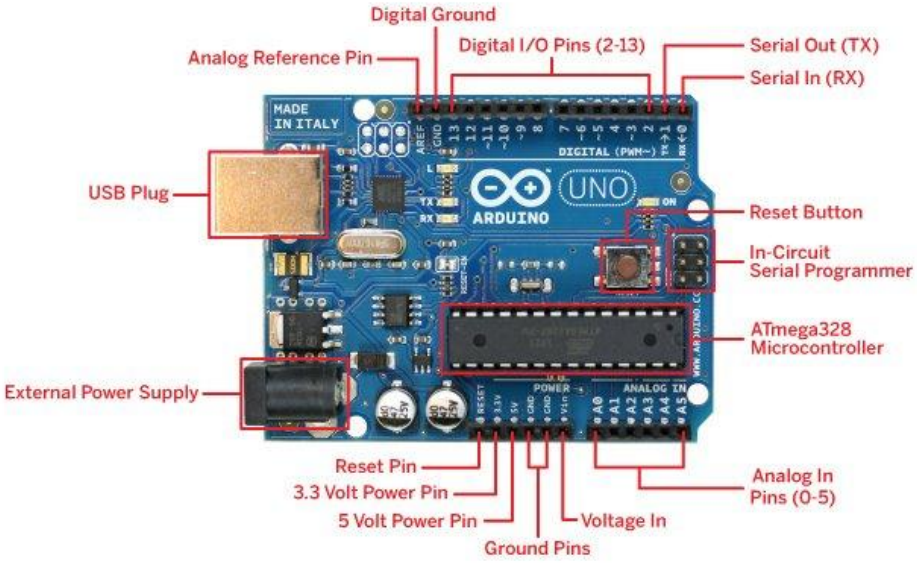


Figure 2.1 The Arduino Uno pinout

2.1.1 The ATmega328 Microcontroller

The ATmega328 is a single chip microcontroller created by Atmel in the mega AVR family. The ATmega328 is a 8-bit AVR RISC-based microcontroller combines 32 KB ISP flash memory with read-while-write capabilities, 1 KB EEPROM, 2 KB SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible timer/counters with compare modes, internal and external interrupts, serial programmable USART, a byte-oriented 2-wire serial interface, SPI serial port, 6-channel 10-bit A/D converter and a programmable watchdog timer with internal oscillator. The device operates between 1.8-5.5 volts. The device achieves throughput approaching 1 MIPS per MHz.

2.1.2 Digital Pins

The digital pins on an Arduino board can be used for general purpose input and output via the pinMode(), digitalRead(), and digitalWrite() commands in the Arduino software IDE. Each pin has an internal pull-up resistor which can be turned on and off using digitalWrite() (w/ a value of HIGH or LOW, respectively) when the pin is configured as an input. The maximum current per pin is 40 mA.

2.1.2.1: Serial: 0 (RX) and 1 (TX)

Used to receive (RX) and transmit (TX) TTL serial data.

2.1.2.2: External Interrupts: 2 and 3

These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

2.1.2.3: PWM: 3, 5, 6, 9, 10, and 11

Provides 8-bit PWM output with the analogWrite() function.

2.1.2.4: SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK)

These pins support SPI communication, which, although provided by the underlying hardware, is not currently included in the Arduino language.

2.1.3 Analog Pins

In addition to the specific functions listed below, the analog input pins support 10-bit analog-to-digital conversion (ADC) using the analogRead() function. Most of the analog inputs can also be used as digital pins: analog input 0 as digital pin 14 through analog input 5 as digital pin 19.

2.1.4 Power Pins

2.1.4.1: VIN (labeled "9V")

It is the input voltage to the Arduino board when it's using an external power source (as opposed to 5 volts from the USB connection or other regulated power source). Voltage may be supplied through this pin, or, if voltage is supplied via the power jack, it may be accessed through this pin.

2.1.4.2: 5V

This is the he regulated power supply used to power the microcontroller and other components on the board. It may come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.

2.1.4.3: 3V3.

It is a 3.3 volt supply generated by the on-board FTDI chip.

2.1.4.4: GND

The board provides two pins as ground pins

2.2 Arduino Software (IDE)

The Arduino project provides the Arduino integrated development environment (IDE), which is a cross-platform application written in the programming language Java. It originated from the IDE for the languages Processing and Wiring. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and provides simple one-click mechanism to compile and load programs to an Arduino board. A program written with the IDE for Arduino is called a "sketch".

The Arduino IDE supports the languages C and C++ using special rules to organize code. The Arduino IDE supplies a software library called Wiring from the Wiring project, which provides many common input and output procedures. A typical Arduino C/C++ sketch consist of two functions that are compiled and linked with a program stub main() into an executable cyclic executive program:

- **setup():** a function that runs once at the start of a program and that can initialize settings.
- **loop():** a function called repeatedly until the board powers off.

After compiling and linking with the GNU toolchain, also included with the IDE distribution, the Arduino IDE employs the program avrdude to convert the executable code into a text file in hexadecimal coding that is loaded into the Arduino board by a loader program in the board's firmware.

The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux.

2.2.1 Features of Arduino IDE

2.2.1.1 Inexpensive

Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled.

2.2.1.2 Cross-platform

The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux and other operating systems. Most microcontroller systems are limited to Windows.

2.2.1.3 Simple, clear programming environment

The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.

2.2.1.4 Open source and extensible software

The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, AVR-C code can be directly added into your Arduino programs.

2.2.1.5 Open source and extensible hardware

The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extend it and improve it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

CHAPTER 3: BLUETOOTH

INTRODUCTION

Bluetooth typically has a range of about 10m so if the user wants to operate home appliances remotely using the android phone the Bluetooth technology may suffice. In our HAS project we have employed the HC05 Bluetooth module to communicate with the smartphone. As the Bluetooth module receives a character from the smartphone, it sends a message to the Arduino uno to make the pin corresponding to the device HIGH.

3.1 THE BLUETOOTH TECHNOLOGY

Bluetooth is a wireless technology standard for exchanging data over short distances (using short-wavelength UHF radio waves in the ISM band from 2.4 to 2.485 GHz) from fixed and mobile devices and building personal area networks (PANs).

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 (FHSS). 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. Bluetooth low energy uses 2 MHz spacing, which accommodates 40 channels.

Bluetooth is a standard wire-replacement communications protocol primarily designed for low-power consumption, with a short range based on low-cost transceiver microchips in each device. Since the devices use a radio (broadcast) communications system, they do not have to be in visual line of sight of each other, however a quasi optical wireless path must be viable. Range is typically 10m although it may be extended to 100m.

3.2 THE BLUETOOTH MODULE HC-05

The HC-05 is serially connected to the Arduino uno board. It is a module that can be set to be either Master or Slave. It has 6 major pins.

3.2.1 The HC-05 pinout

3.2.1.1 KEY

If brought HIGH before power is applied, forces AT command Setup Mode and the LED blinks slowly i.e. one every two seconds.

3.2.1.2 VCC

Receives +5V power from the Arduino board.

3.2.1.3 GND

System/Arduino ground

3.2.1.4 TXD

Transmits serial data from HC-05 to Arduino receive.

3.2.1.5 RXD

Receives serial data from Arduino uno transmit

3.2.1.6 STATE

Tells if the device is connected or not

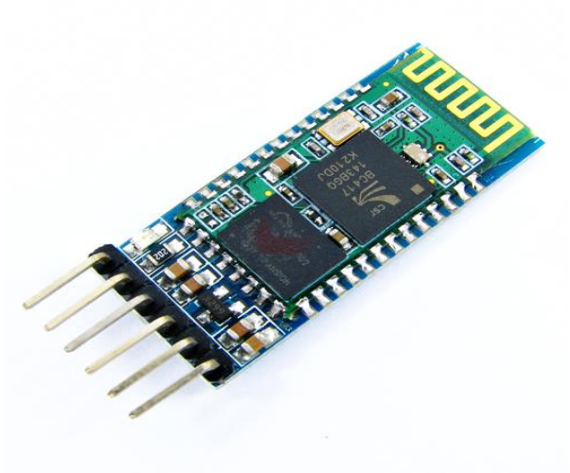


Figure 3.1: Bluetooth Module

3.3 MODES OF OPERATION OF HC-05

HC-05 embedded Bluetooth serial communication module (can be short for module) has two work modes: order-response work mode (AT mode) and automatic connection work mode (Command Mode). And there are three work roles (Master, Slave and Loopback) at the automatic connection work mode. When the module is at the automatic connection work mode, it will follow the default way set lastly to transmit the data automatically. When the module is at the order-response work mode, user can send the AT command to the module to set the control parameters and sent control order. The work mode of module can be switched by controlling the module PIN (PIO11) input level.

3.3.1 Serial module PINs:

3.3.1.1 PIO8 connects with LED. When the module is power on, LED will flicker. And the flicker style will indicate which work mode is in using since different mode has different flicker time interval.

3.3.1.2 PIO9 connects with LED. It indicates whether the connection is built or not. When the Bluetooth serial is paired, the LED will be turned on. It means the connection is built successfully.

3.3.1.3 PIO11 is the work mode switch. When this PIN port is input high level, the work mode will become order-response work mode. While this PIN port is input low level or suspended in air, the work mode will become automatic connection work mode.

3.3.1.4 The module can be reset if it is re-powered since there is a reset circuit at the module.

3.3.2 Getting to the AT mode and Command Mode

Firstly, we apply input low level to KEY. Then we supply power to the module. Next, we input high level to the KEY. Then the module will enter to AT mode. The baud rate is as same as the communication time, such as 9600 etc. To get to the command mode, we input low level to KEY

and supply power to the module. Then the module will enter to command mode. It can be used for pairing

3.4 AT commands

AT commands are case-sensitive and should end up with terminator (“enter” or “\r\n”)

Function	Command	Response	Parameter
Test	AT	OK	None
Reset	AT+RESET	OK	None
Get the software version	AT+VERSION?	+Version: <Param> OK	Param: Version number
Restore default status	AT+ORGL	OK	None
Get the Module Address	AT+ADDR?	+ADDR: <Param> OK	Param: Bluetooth address
Set/Inquire devices’s name	AT+NAME=<Param>	OK	Param: Bluetooth device name Default: “HC-05”
	AT+name?	1. +NAME:<Param> OK----success 2. FAIL---failure	
Set/Inquire module role	AT+ROLE=<Param>	OK	Param: 0---slave 1----master 2----Slave-Loop Default:0
	AT+ROLE?	+ROLE:<Param> OK	
Set/Inquire passkey	AT+PSWD=<param>	OK	Param: passkey Default:”1234”
	AT+PSWD?	+PSWD: <Param>	

		OK	

Table 3.4: AT commands

The above AT command modes may be used to personalize the Bluetooth module in terms of the desired baud rate, module name and module password.

3.5 INTERFACING THE BLUETOOTH MODULE AND ARDUINO

Only four connections are made between the Bluetooth module and the Arduino. The VCC and GND of the Bluetooth module are connected to the VCC and GND of the Arduino UNO respectively. The TXD of the BT module is connected to RX of Arduino and the RXD of BT module is connected to the TX of the Arduino.

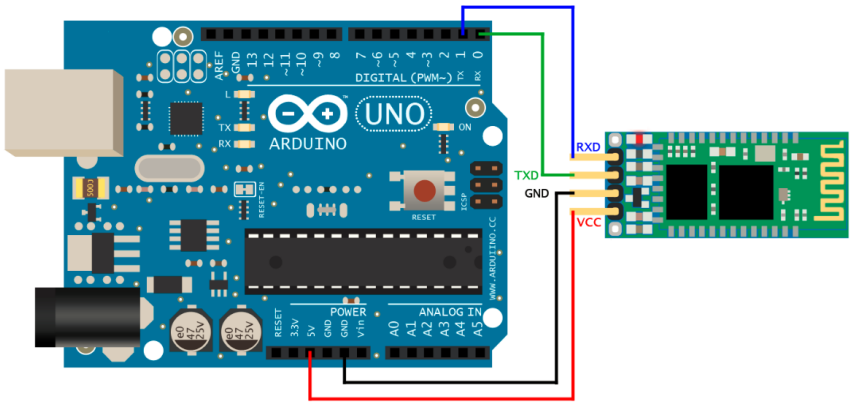


Figure 3.2: Interfacing BT-module and Arduino

CHAPTER 4: GSM

INTRODUCTION

The GSM technology and the GSM module(SIM900) makes it possible for the user to control the appliances remotely outside the home where GSM network is available. It operates based on the concept of an SMS which is sent to the GSM module from the Android smartphone. When the module receives the SMS it communicates to the Arduino uno board via the TX and RX pins and the corresponding pin of the device is made HIGH.

4.1 THE GSM(SIM900) BOARD

GSM (Global System for Mobile) / GPRS (General Packet Radio Service) TTL –Modem is SIM900 Quad-band GSM / GPRS device, works on frequencies 850 MHZ, 900 MHZ, 1800 MHZ and 1900 MHZ. It is very compact in size and easy to use as plug in GSM Modem. The Modem is designed with 3V3 and 5V DC TTL interfacing circuitry, which allows User to directly interface with 5V Microcontrollers (PIC, AVR, Arduino, 8051, etc.) as well as 3V3 Microcontrollers (ARM, ARM Cortex XX, etc.). The baud rate can be configurable from 9600- 115200 bps through AT (Attention) commands. This GSM/GPRS TTL Modem has internal TCP/IP stack to enable User to connect with internet through GPRS feature. It is suitable for SMS as well as DATA transfer application in mobile phone to mobile phone interface. The modem can be interfaced with a Microcontroller using USART (Universal Synchronous Asynchronous Receiver and Transmitter) feature (serial communication).

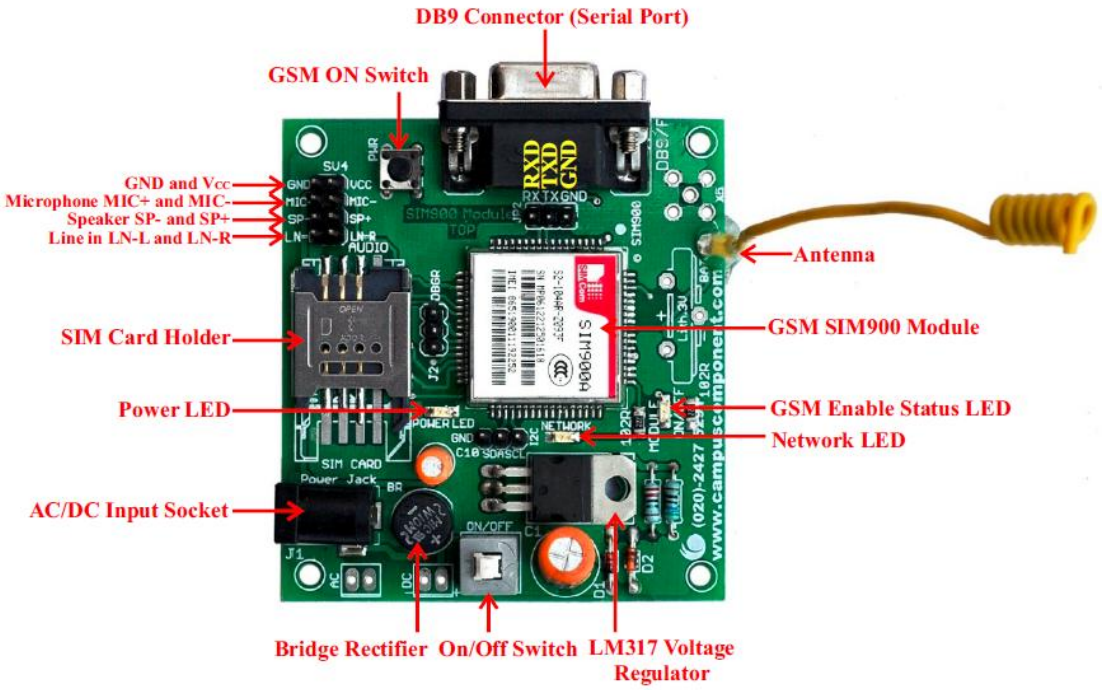


Figure 4.1: Pinout of GSM module

SIM900A GSM Module:

This is actual SIM900 GSM module which is manufactured by SIMCom. Designed for global market, SIM900 is a quad-band GSM/GPRS engine that works on frequencies GSM 850MHz, EGSM 900MHz, DCS 1800MHz and PCS 1900MHz. SIM900 features GPRS multislot class 10/ class 8 (optional) and supports the GPRS coding schemes CS-1, CS-2, CS-3 and CS-4. With a tiny configuration of 24mm x 24mm x 3mm, SIM900 can meet almost all the space requirements in User's applications, such as M2M, smart phone, PDA and other mobile devices.

MAX232 IC:

The MAX232 is an integrated circuit that converts signals from an RS-232 serial port to signals suitable for use in TTL compatible digital logic circuits, so that devices works on TTL logic can share the data with devices connected through Serial port (DB9 Connector).

Serial port / DB9 connector:

User just needs to attach RS232 cable here so that it can be connected to a device which has Serial port / DB9 Connector.

Power Supply Socket:

This power supply socket which actually named as AC/DC Socket provides the functionality to user to connect external power supply from Transformer, Battery or Adapter through DC jack. User can provide maximum of 12V AC/DC power supply through AC/DC socket. This is power supply designed into maximum protection consideration so that it can even prevent reverse polarity DC power supply as well as DC conversion from AC power Supply. It also includes LM317 Voltage Regulator which provides an output voltage adjustable over a1.2V to 37V.

Power On/Off and GSM ON Switch:

Power On/Off switch is type of push-on push-off DPDT switch which is used for only make power supply on/off provided through AC/DC Socket indicated by 'Power LED'. GSM ON Switch is type of Push on DPST tactile switch which is used for only to make GSM module 'On' indicated by 'Module On/Off LED' while initiating with Network indicated by 'Network Indication LED'.

SIM (Subscriber Identity Module) Card Slot:

This onboard SIM card slot provide User functionality of insert a SIM (GSM only) card of any service provider.

While inserting in and removing out SIM card from SIM card slot, User must take precaution that power supply should be OFF so that after making Power supply ON it will be easy to reinitialize with SIM for this module.

Indicator LEDs:

Indicator LEDs just used to indicate status accordingly. These are three LEDs represent

Power On/Off Status, Network Status and Module On/Off Status respectively. Power LED will keep on until the power supply is enabling to this board by using push-on push-off switch. Network Status LED will show whether inserted SIM card successfully connected to service provider's network or not, in short signal strength. Module On/Off indicator LED will show status of GSM module's power on/off.

GSM Interfacing Board

RXD, TXD and GND pins (JP2):

These pins are used to connect devices which need to be connected to GSM module through USART (Universal Synchronous Asynchronous Receiver and Transmitter) communication. Devices

may be like Desktop or Laptop Computer System, Microcontrollers, etc. RXD (Receive Data) should be connected to TXD (Transmit Data) of other device and vice versa, whereas GND (Ground) should be connected to other device's GND pin to make ground common for both systems.

Audio Connectors:

Audio Connectors deals with Audio related operations. These are eight pins in a group of two each denoted by **SV4**. GND (0V Supply) and VCC (+5V Supply) are used to have source for external device. MIC+ and module.

Debugger (DBG-R and DBG-T) Connectors (J2):

These connectors are 2-wire null modem interface DBG_TXD and DBG_RXD. These pins can be used for debugging and upgrading firmware. User generally doesn't need to deal with these pins.

4.2 FEATURES OF THE BOARD

· Quad Band GSM/GPRS : 850 / 900 / 1800 / 1900 MHz
· Built in RS232 to TTL or viceversa Logic Converter (MAX232)
· Configurable Baud Rate
· SMA (SubMiniature version A) connector with GSM L Type Antenna
· Built in SIM (Subscriber Identity Module) Card holder
· Built in Network Status LED
· Inbuilt Powerful TCP / IP (Transfer Control Protocol / Internet Protocol) stack for internet data transfer through GPRS (General Packet Radio Service)
· Audio Interface Connectors (Audio in and Audio out)
· Most Status and Controlling pins are available
· Normal Operation Temperature : -20 °C to +55 °C
· Input Voltage : 5V to 12V DC
· LDB9 connector (Serial Port) provided for easy interfacing

4.3 AT COMMANDS FOR GSM MODULE

Command	Description	Return	Result
AT+CNMI	Set the new message remind, for example AT+CNMI=2,1,	+CMTI:"SM",2	when set is on and message box is NOT full, message is stored at position 2
AT+CMGF	Set the module message mode, set either at PDU(0) or text mode(1)	OK	
AT+CSCS	Set TE character set, set AT+CSCS="GSM" for english only message, or set AT+CSCS="UCS2" for other language	OK	
AT+CMGD	delete message, to delete message at postion 1: AT+CMGD=1	OK	
AT+CMGR	read message, for exmple, AT+CMGR=1 to read message at position 1	-	
AT+CPMS	inquiry or set message storage settings	AT+CPMS? to check how many message can be stored maximally and how many message stored	: +CPMS:"SM",1,50, means support 50 sms max and 1 sms are stored now
AT+CMGS	send message, send 180 bytes at GSM mode, or 70 Chinese character at UCS2 mode, AT+CMGS="18576608994"	will return ">" and then type message, then end up with hex value 1A(0X1A, "CTRL+Z"), send 1B to cancel "ESC"	and finally return: +CMGS:156, in which 156 has meaning.

4.4 INTERFACING THE GSM MODULE AND ARDUINO UNO

The GSM module is simply interfaced with the Arduino uno board on connecting the TX pin of GSM to RX of Arduino and the RX pin of GSM to the TX of the Arduino. External power must be supplied to the GSM module via a 12V adapter.

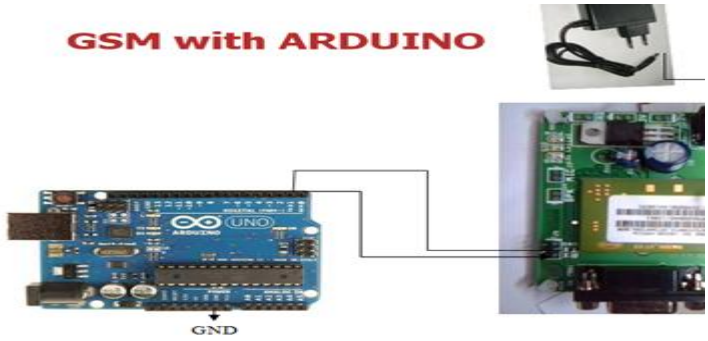


Figure 4.2: Interfacing The GSM module and the Arduino

CHAPTER 5: ETHERNET

INTRODUCTION

The Ethernet shield used in our project serves the purpose of making the HAS internet-enabled. For each appliance a HTML page is created to control the appliances remotely using the internet.

5.1 THE ETHERNET SHIELD

The Arduino Ethernet Shield allows easy connection of Arduino to the internet. This shield enables the Arduino to send and receive data from anywhere in the world with an internet connection.
The module requires plug in onto the Arduino board and connection to a network with an RJ45 cable. It requires the following:

- Requires an Arduino board (not included)
- Operating voltage 5V (supplied from the Arduino Board)
- Ethernet Controller: W5100 with internal 16K buffer
- Connection speed: 10/100Mb
- Connection with Arduino on SPI port

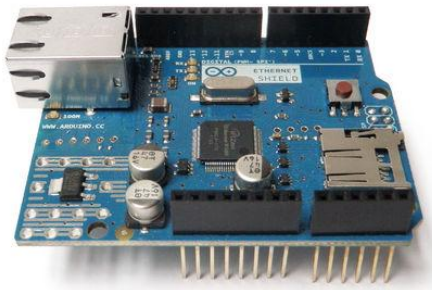


Figure 5.1: The Ethernet Shield

The Arduino Ethernet Shield allows an Arduino board to connect to the internet. It is based on the Wiznet W5100 ethernet chip. The Wiznet W5100 provides a network (IP) stack capable of both TCP and UDP. It supports up to four simultaneous socket connections. We use the Ethernet library to write sketches which connects to the internet using the shield. The ethernet shield

connects to an Arduino board using long wire-wrap headers which extend through the shield. This keeps the pin layout intact and allows another shield to be stacked on top.
The Ethernet Shield has a standard RJ-45 connection, with an integrated line transformer and Power over Ethernet enabled.
There is an onboard micro-SD card slot, which can be used to store files for serving over the network. It is compatible with all the Arduino/Genuino boards. The on-board micro SD card reader is accessible through the SD Library. When working with this library, SS is on Pin 4. The original revision of the shield contained a full-size SD card slot; this is not supported.
The shield also includes a reset controller, to ensure that the W5100 Ethernet module is properly reset on power-up. Previous revisions of the shield were not compatible with the Mega and need to be manually reset after power-up.

The current shield has a Power over Ethernet (PoE) module designed to extract power from a conventional twisted pair Category 5 Ethernet cable:

- IEEE802.3af compliant
- Low output ripple and noise (100mVpp)
- Input voltage range 36V to 57V
- Overload and short-circuit protection
- 9V Output
- High efficiency DC/DC converter: typ 75% @ 50% load
- 1500V isolation (input to output)

The shield does not come with the PoE module built in, it is a separate component that must be added on.
Arduino communicates with both the W5100 and SD card using the SPI bus (through the ICSP header). This is on digital pins 10, 11, 12, and 13 on the Uno and pins 50, 51, and 52 on the Mega. On both boards, pin 10 is used to select the W5100 and pin 4 for the SD card. These pins cannot be used for general I/O. On the Mega, the hardware SS pin, 53, is not used to select either the W5100 or the SD card, but it must be kept as an output or the SPI interface won't work.
Note that because the W5100 and SD card share the SPI bus, only one can be active at a time. If you are using both peripherals in your program, this should be taken care of by the corresponding libraries. If you're not using one of the peripherals in your program, however, you'll need to explicitly deselect it. To do this with the SD card, set pin 4 as an output and write a high to it. For the W5100, set digital pin 10 as a high output.

The shield provides a standard RJ45 ethernet jack.

The reset button on the shield resets both the W5100 and the Arduino board.

The shield contains a number of informational LEDs:

- PWR: indicates that the board and shield are powered
- LINK: indicates the presence of a network link and flashes when the shield transmits or receives data
- FULLD: indicates that the network connection is full duplex
- 100M: indicates the presence of a 100 Mb/s network connection (as opposed to 10 Mb/s)
- RX: flashes when the shield receives data
- TX: flashes when the shield sends data
- COLL: flashes when network collisions are detected

The solder jumper marked "INT" can be connected to allow the Arduino board to receive interrupt-driven notification of events from the W5100, but this is not supported by the Ethernet library. The jumper connects the INT pin of the W5100 to digital pin 2 of the Arduino.

CHAPTER 6: THE ANDROID APP

INTRODUCTION

The software we chose for the designing of our app is the MIT app inventor 2 which is free online software aimed at building applications for android. App Inventor for Android is an open-source web application originally provided by Google, and now maintained by the Massachusetts Institute of Technology (MIT).

It allows newcomers to computer programming to create software applications for the Android operating system (OS). It uses a graphical interface, very similar to Scratch and the StarLogo TNG user interface, which allows users to drag-and-drop visual objects to create an application that can run on Android devices. In creating App Inventor, Google drew upon significant prior research in educational computing, as well as work done within Google on online development environments.

App Inventor and the projects on which it is based are informed by constructionist learning theories, which emphasizes that programming can be a vehicle for engaging powerful ideas through active learning. As such, it is part of an ongoing movement in computers and education that began with the work of Seymour Papert and the MIT Logo Group in the 1960s and has also manifested itself with Mitchel Resnick's work on Lego Mindstorms and StarLogo.

6.1 HAS APP PAGE 1 BLUETOOTH

The page 1 for the HAS app is Bluetooth in which the user is prompted to connect to the Bluetooth module HC-05. Also the home appliances such light, fan etc., are listed which their corresponding ON and OFF switch.

For Bluetooth communication the MIT app inventor offers a component named **BluetoothClient** which is added to the Bluetooth page. The BluetoothClient component consists of the following:

Properties

AddressesAndNames

The addresses and names of paired Bluetooth devices

Available

Whether Bluetooth is available on the device

CharacterEncoding

DelimiterByte

Enabled

Whether Bluetooth is enabled

HighByteFirst

IsConnected

Secure

Whether to invoke SSP (Simple Secure Pairing), which is supported on devices with Bluetooth v2.1 or higher. When working with embedded Bluetooth devices, this property may need to be set to False. For Android 2.0-2.2, this property setting will be ignored.

Events

none

Methods

number BytesAvailableToReceive()

Returns an estimate of the number of bytes that can be received without blocking

boolean Connect(text address)

Connect to the Bluetooth device with the specified address and the Serial Port Profile (SPP). Returns true if the connection was successful.

boolean ConnectWithUUID(text address, text uuid)

Connect to the Bluetooth device with the specified address and UUID. Returns true if the connection was successful.

Disconnect()

Disconnect from the connected Bluetooth device.

boolean IsDevicePaired(text address)

Checks whether the Bluetooth device with the specified address is paired.

number ReceiveSigned1ByteNumber()

Receive a signed 1-byte number from the connected Bluetooth device.

number ReceiveSigned2ByteNumber()

Receive a signed 2-byte number from the connected Bluetooth device.

number ReceiveSigned4ByteNumber()

Receive a signed 4-byte number from the connected Bluetooth device.

list ReceiveSignedBytes(number numberOfBytes)

Receive multiple signed byte values from the connected Bluetooth device. If numberOfBytes is less than 0, read until a delimiter byte value is received.

text ReceiveText(number numberOfBytes)

Receive text from the connected Bluetooth device. If numberOfBytes is less than 0, read until a delimiter byte value is received.

number ReceiveUnsigned1ByteNumber()

Receive an unsigned 1-byte number from the connected Bluetooth device.

number ReceiveUnsigned2ByteNumber()

Receive a unsigned 2-byte number from the connected Bluetooth device.

number ReceiveUnsigned4ByteNumber()

Receive a unsigned 4-byte number from the connected Bluetooth device.

list ReceiveUnsignedBytes(number numberOfBytes)

Receive multiple unsigned byte values from the connected Bluetooth device. If numberOfBytes is less than 0, read until a delimiter byte value is received.

Send1ByteNumber(text number)

Send a 1-byte number to the connected Bluetooth device.

Send2ByteNumber(text number)

Send a 2-byte number to the connected Bluetooth device.

Send4ByteNumber(text number)

Send a 4-byte number to the connected Bluetooth device.

SendBytes(list list)

Send a list of byte values to the connected Bluetooth device.

SendText(text text)

Send text to the connected Bluetooth device.

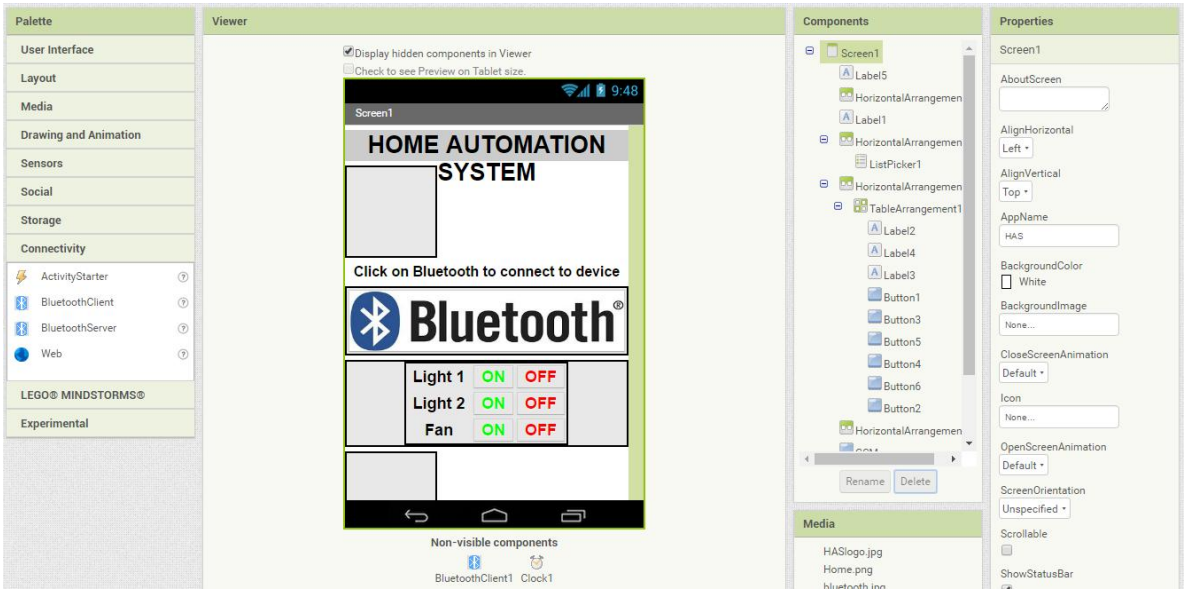


Figure 6.1: Bluetooth Screen 1

Under the tab sensors we need to add another component- **clock** which is a non-visible component that provides the instant in time using the internal clock on the phone. It can fire a timer at regularly set intervals and perform time calculations, manipulations, and conversions.

On clicking ON for light 1 a character "A" is sent from the mobile phone to the Bluetooth module. The BT-module then sends a signal to Arduino indicating the pin for light1 must be HIGH.

The code for each page of the app is to be written in blocks which are discussed in chapter 8.

6.2 HAS APP PAGE 2 GSM

A new screen must be added to the project to obtain the page 2 for the app. This requires one non-visible component – **texting.**

A component that will, when the SendMessagemethod is called, send the text message specified in the Message property to the phone number specified in the PhoneNumberproperty.

If the ReceivingEnabled property is set to 1 messages will not be received. IfReceivingEnabled is set to 2 messages will be received only when the application is running. Finally if ReceivingEnabled is set to 3, messages will be received when the application is running and when the application is not running they will be queued and a notification displayed to the user.

When a message arrives, theMessageReceived event is raised and provides the sending number and message.

An app that includes this component will receive messages even when it is in the background (i.e. when it is not visible on the screen) and, more so, even if the app is not running, so long as it is installed on the phone. If the phone receives a text message when the app is not in the foreground, the phone will show a notification in the notification bar. Selecting the notification will bring up the app. As an app developer, you will probably want to give your users the ability to control ReceivingEnabled so that they can make the phone ignore text messages.

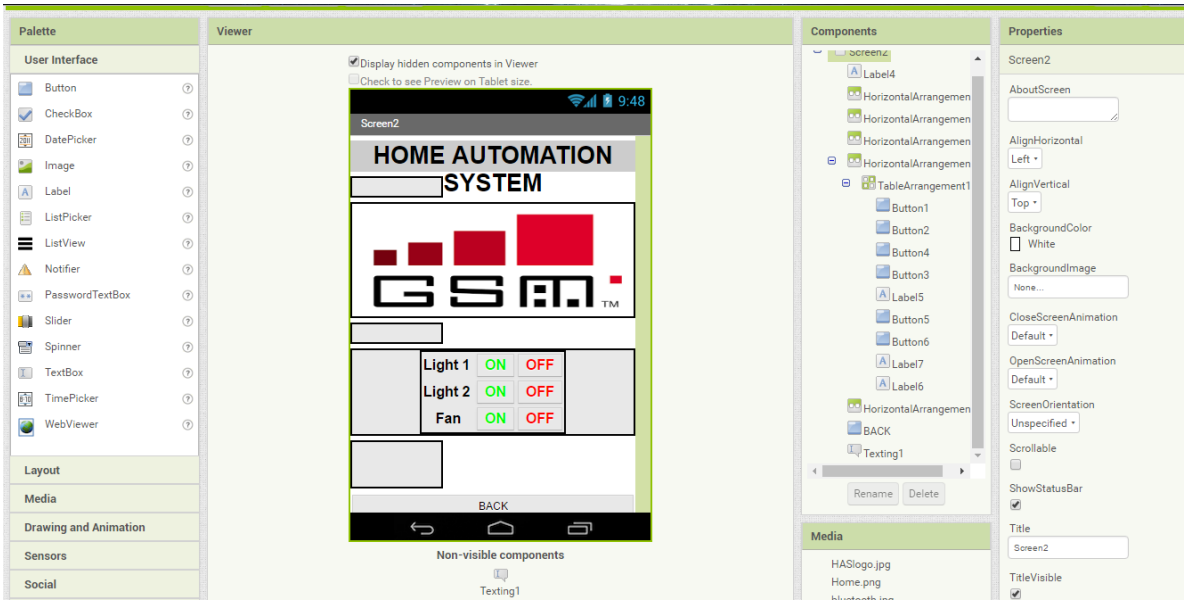


Figure 6.2: GSM screen 2

Similar to the page 1, the GSM page will also have the list of home appliances with their corresponding ON and OFF buttons. On clicking ON for light1, text message #LIGHT a1 is sent to the phone number of the SIM card inserted in the GSM module. The GSM module then communicates to the Arduino to make the corresponding pin of the appliance HIGH.

The code is discussed in chapter 8.

6.3 HAS APP PAGE 3 ETHERNET

For the Ethernet page two non visible components are added

-Web1

It is a non-visible component that provides functions for HTTP GET, POST, PUT, and DELETE requests.

Properties

AllowCookies

Whether the cookies from a response should be saved and used in subsequent requests. Cookies are only supported on Android version 2.3 or greater.

RequestHeaders

The request headers, as a list of two-element sublists. The first element of each sublist represents the request header field name. The second element of each sublist represents the request header field values, either a single value or a list containing multiple values.

ResponseFileName

The name of the file where the response should be saved. If SaveResponse is true and ResponseFileName is empty, then a new file name will be generated.

SaveResponse

Whether the response should be saved in a file.

Url

The URL for the web request.

Events

GotFile(text url, number responseCode, text responseType, text fileName)

Event indicating that a request has finished.

GotText(text url, number responseCode, text responseType, text responseContent)

Event indicating that a request has finished.

Methods

text BuildRequestData(list list)

Converts a list of two-element sublists, representing name and value pairs, to a string formatted as application/x-www-form-urlencoded media type, suitable to pass to PostText.

ClearCookies()

Clears all cookies for this Web component.

Delete()

Performs an HTTP DELETE request using the Url property and retrieves the response. If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. If the SaveResponse property is false, the GotText event will be triggered.

Get()

Performs an HTTP GET request using the Url property and retrieves the response. If the SaveResponse property is true, the response will be saved in a file and the GotFile event will be triggered. If the SaveResponse property is false, the GotText event will be triggered.

-ActivityStarter1

It is a component that can launch an activity using the StartActivity method.

Activities that can be launched include:

- Starting another App Inventor for Android app.
- Starting the camera application by setting the following properties:
    - Action: android.intent.action.MAIN
    - ActivityPackage: com.android.camera
    - ActivityClass: com.android.camera.Camera
- Performing web search.
- Opening a browser to a specified web page.

Properties

Action

ActivityClass

ActivityPackage

DataType

DataUri

Extras

Accepts a list of pairs which are used as key/value pairs in the "Extra" field of the activity

ExtraKey*(Obsolete)*

ExtraValue*(Obsolete)*

Result

ResultName

ResultType

ResultUri

Events

AfterActivity(text result)

Event raised after this ActivityStarter returns.

ActivityCanceled()

Event raised if this ActivityStarter returns because the activity was canceled.

Methods

text ResolveActivity()

Returns the name of the activity that corresponds to this ActivityStarter, or an empty string if no corresponding activity can be found.

StartActivity()

Start the activity corresponding to this ActivityStarter.

For the control of each appliance, a HTML page is created. For example, on clicking ON for light1 the aim of the page is to direct the user to the corresponding webpage. The device can now be operated online.

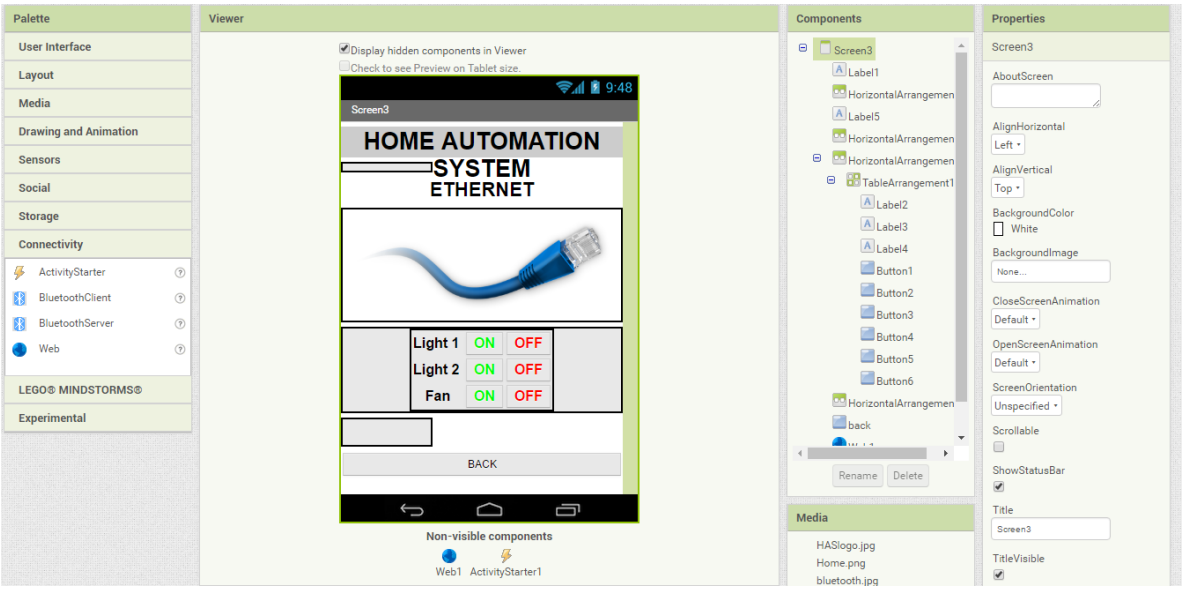


Figure 6.3: Ethernet screen 3

CHAPTER 7: RELAYS

INTRODUCTION

Relays have been used as an electric switch which on receiving a HIGH voltage allows the device to be in the ON mode. A separate relay has been employed to each appliance in this project.

7.1 DEFINING RELAYS

Relays are simple switches which are operated both electrically and mechanically. Relays consist of a n electromagnet and also a set of contacts. The switching mechanism is carried out with the help of the electromagnet. There are also other operating principles for its working. But they differ according to their applications. Most of the devices have the application of relays.

Relays have the exact working of a switch. So, the same concept is also applied. A relay is said to switch one or more poles. Each pole has contacts that can be thrown in mainly three ways. They are

- **Normally Open Contact (NO)** – NO contact is also called a make contact. It closes the circuit when the relay is activated. It disconnects the circuit when the relay is inactive.
- **Normally Closed Contact (NC)** – NC contact is also known as break contact. This is opposite to the NO contact. When the relay is activated, the circuit disconnects. When the relay is deactivated, the circuit connects.
- **Change-over (CO) / Double-throw (DT) Contacts** – This type of contacts are used to control two types of circuits. They are used to control a NO contact and also a NC contact with a common terminal. According to their type they are called by the names **break before make** and **make before break** contacts.
-

7.2 RELAY DESIGN

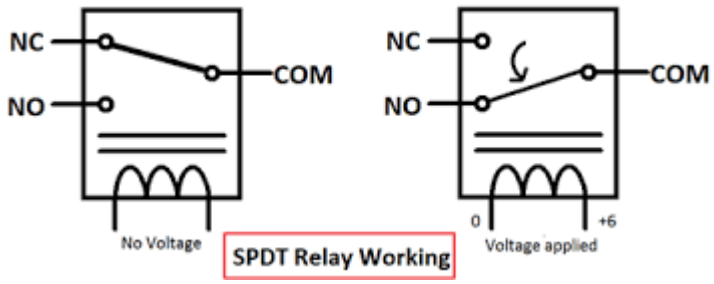


Figure 7.1: Working of relays

When voltage is applied, it activates the electromagnet, generating a magnetic field that attracts a contact and activates the device connected to the relay. When the power is switched off, a spring pulls the contact back up to its original position, switching circuit off again.
A "normally open" (NO) relay: the contacts in the figure 7.1(b) are not connected by default, and it switches on only when a current flows through the magnet. Other relays are "normally closed" (NC; the contacts are connected so a current flows through them by default) and switch off only when the magnet is activated, pulling or pushing the contacts apart. Normally open relays are the most common

.

CHAPTER 8: CODE

INTRODUCTION

This chapter includes the coding used in for each component used in the HAS project.

8.1 CODE FOR ARDUINO UNO

The code for Arduino includes the code for BT module, GSM module and Ethernet Shield. This code must be uploaded to the Arduino uno board.

```
#include <SoftwareSerial.h>
#include <EEPROM.h>
#include <Servo.h>
#include <Wire.h>
#include <Teleduino328.h>
#include <SPI.h>
#include <Ethernet.h>

// User configurable variables
byte useDhcp = false;
byte useDns = true;
byte mac[] = { 0xDE, 0xFD, 0xFE, 0x6F, 0xFE, 0x10 };
IPAddress deviceIp(14, 139, 240, 60); // Only if useDhcp is false
IPAddress gatewayIp(14, 139, 240, 49); // Only if useDhcp is false
IPAddress dnsIp(14, 139, 5, 5); // Only if useDhcp is false
IPAddress subnet(255, 255, 255, 248); // Only if useDhcp is false
IPAddress serverIp(173, 230, 152, 173); // Only if useDns is false
char serverName[] = "us01.proxy.teleduino.org"; // Only if useDns is true
unsigned int serverPort = 5353; // Can be set to either 53 or 5353
byte statusLedPin = 8;

byte key[] = { 0xA3, 0x61, 0x07, 0x1E,
               0x57, 0x8B, 0xC6, 0x0E,
               0x2A, 0x97, 0xA1, 0x48,
               0x5D, 0x72, 0x49, 0x57 };
// Other required variables
byte data[257];
byte dataLength;
byte hexStage;
unsigned long lastInstruction = 0;
unsigned long lastRefresh = 0;
byte stage = 0;

// Declare client object
EthernetClient Client;
```

```
char inchar; // Will hold the incoming character from the GSM shield
SoftwareSerial SIM900(7, 8);
char val; // variable to receive data from the serial port
int l1 = 9; // LED connected to pin 48 (on-board LED)
int l2=10;
int l3=11;
void setup() {
  Serial.begin(9600);
 pinMode(l1, OUTPUT);
 pinMode(l2, OUTPUT);
 pinMode(l3, OUTPUT);
 digitalWrite(l1, LOW);
 digitalWrite(l2, LOW);
 digitalWrite(l3, LOW);
 SIM900.begin(9600);
 SIM900.print("AT+CMGF=1\r");      // set SMS mode to text
 delay(100);
 SIM900.print("AT+CNMI=2,2,0,0,0\r");
 delay(100);
 Serial.println("Ready...");
// Load presets
 Teleduino328.loadPresets();

 // Set status LED pin

Teleduino328.setStatusLedPin(statusLedPin);
 Teleduino328.setStatusLed(1);                // Initialisation
 // Check the EEPROM header and check to see if the key is correct
 // This is to ensure the key is not cleared from the EEPROM
 if(EEPROM.read(0) != '#')
 {
  EEPROM.write(0, '#');
 }
 if(EEPROM.read(1) != 0)
 {
  EEPROM.write(1, 0);
 }
 if(EEPROM.read(2) != '#')
 {
  EEPROM.write(2, '#');
 }
 if(EEPROM.read(160) != '#')
 {
  EEPROM.write(160, '#');
 }
 for(byte i = 0; i < 16; i++)
 {
  if(EEPROM.read(161 + i) != key[i])
  {
    EEPROM.write(161 + i, key[i]);
  }
 }
 if(EEPROM.read(177) != '#')
 {
  EEPROM.write(177, '#');
 }
```

```
 // Start network and attempt to connect to proxy server
 Teleduino328.setStatusLed(2); // Network configuration
 if(useDhcp)
 {
  if(!Ethernet.begin(mac))
  {
   Teleduino328.setStatusLed(2, false, 10000);
   Teleduino328.reset();
  }
 }
 else
 {
  Ethernet.begin(mac, deviceIp, dnsIp, gatewayIp, subnet);
 }
 delay(1000);

 Teleduino328.setStatusLed(3); // Connect to server
 if((useDns && !Client.connect(serverName, serverPort)) || (!useDns && !Client.connect(serverIp, serverPort)))
 {
  Teleduino328.setStatusLed(3, false, 10000);
  Teleduino328.reset();
 }
 lastInstruction = millis();
}

//void SIM900power()
// software equivalent of pressing the GSM shield "power" button
/*{
 digitalWrite(9, HIGH);
 delay(1000);
 digitalWrite(9, LOW);
 delay(7000);
}*/


void loop() {
 if( Serial.available() )       // if data is available to read
 {
  val = Serial.read();         // read it and store it in 'val'
  Serial.println(val);

 if( val == 'A' )
 {
  digitalWrite(l1, HIGH);  // turn ON the LED
 }
 else if(val == 'B')
 {
  digitalWrite(l1, LOW);   // otherwise turn it OFF
 }
 else if(val == 'C')
 {
```

```
  digitalWrite(l2, HIGH);
 }
 else if(val == 'D')
 {
  digitalWrite(l2, LOW);
 }
 else if(val == 'E')
 {
  digitalWrite(l3, HIGH);    // otherwise turn
it OFF
 }
 else if(val == 'F')
 {
  digitalWrite(l3, LOW);
 }
 }
 if(SIM900.available() >0)
 {
  inchar=SIM900.read();
  if (inchar=='#')
  {
   delay(10);

   inchar=SIM900.read();
   if (inchar=='a')
   {
    delay(10);
    inchar=SIM900.read();
    if (inchar=='0')
    {
     digitalWrite(l1, LOW);
    }
    else if (inchar=='1')
{
 digitalWrite(l1, HIGH);
}
 delay(10);
inchar=SIM900.read();
if (inchar=='b')
{
 inchar=SIM900.read();
 if (inchar=='0')
 {
  digitalWrite(l2, LOW);
 }
 else if (inchar=='1')
 {
  digitalWrite(l2, HIGH);
 }
 delay(10);
 inchar=SIM900.read();
 if (inchar=='c')
 {
  inchar=SIM900.read();
  if (inchar=='0')
  {
   digitalWrite(l3, LOW);
  }
  else if (inchar=='1')
  {
   digitalWrite(l3, HIGH);
  }
  delay(10);

 }
```

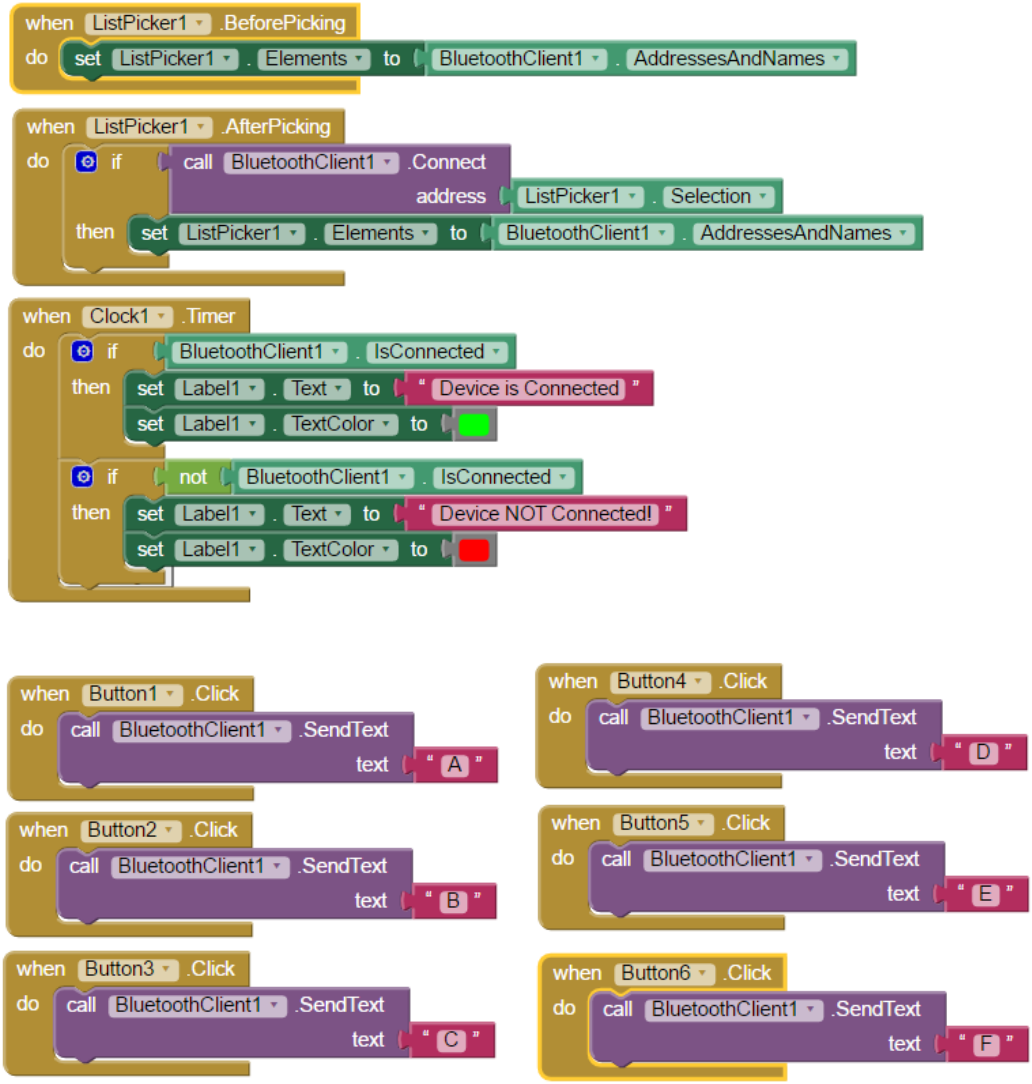
```
        SIM900.println("AT+CMGD=1,4");   //
delete all SMS
       }
     }
   }

}
if(Client.connected())
  {
   // What we need to do depends on which
'stage' we are at
   switch(stage)
   {
    case 0: // Wait for start byte
     if(Client.available())
     {
      char c = Client.read();
      if(c == '?')
      {
       stage++;
      }
     }
     break;
    case 1: // Reset variables
     dataLength = 0;
     hexStage = 0;
     stage++;
     break;
    case 2: // Instruction byte
     if(Client.available())
     {
      char c = Client.read();
      if(c == '?')
      {
       stage = 1;
       break;
      }
      else if(c == '\r' || c == '\n' || c == '.')
      {
       stage = 0;
       break;
      }
      if(!hexStage)
      {
       data[0] = Teleduino328.hexDecode(c)
* 16;
      }
      else
      {
       data[0]                              +=
Teleduino328.hexDecode(c);
      }
      hexStage = !hexStage;
      if(!hexStage)
      {
       stage++;
      }
     }
     break;
    case 3: // Data length byte
     if(Client.available())
     {
      char c = Client.read();
      if(c == '?')
      {
       stage = 1;
```

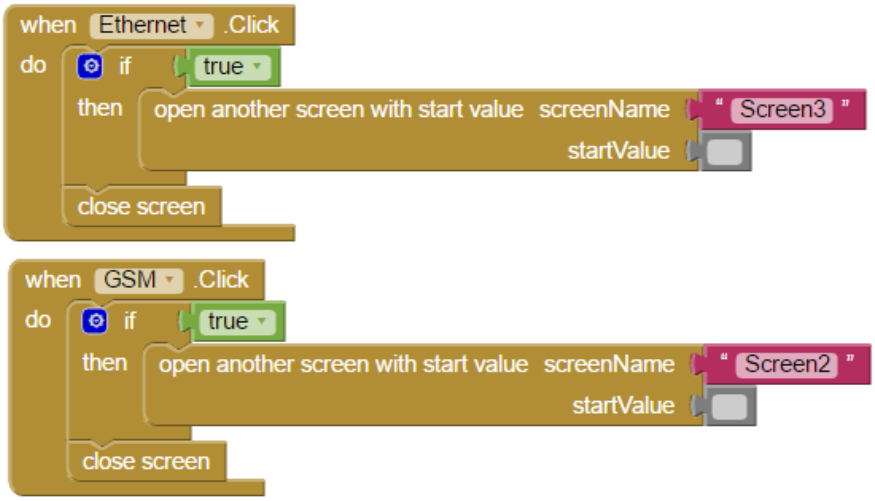
```
          break;
        }
        else if(c == '\r' || c == '\n' || c == '.')
        {
         stage = 0;
         break;
        }
        if(!hexStage)
        {
         data[1] = Teleduino328.hexDecode(c)
* 16;
        }
        else
        {
         data[1]                                    +=
Teleduino328.hexDecode(c);
        }
        hexStage = !hexStage;
        if(!hexStage)
        {
         stage++;
        }
       }
       break;
      case 4: // Data
       if(Client.available())
       {
        char c = Client.read();
        if(c == '?')
        {
         stage = 1;
         break;
        }
        else if(c == '\r' || c == '\n' || c == '.')
        {
         if(dataLength == data[1])
         {
          stage++;
          break;
         }
         else
         {
          stage = 0;
          break;
         }
        }
        if(!hexStage)
        {
         data[2        +        dataLength]        =
Teleduino328.hexDecode(c) * 16;
        }
        else
        {
         data[2        +        dataLength]        +=
Teleduino328.hexDecode(c);
        }
        hexStage = !hexStage;
        if(!hexStage)
        {
         dataLength++;
        }
       }
       break;
      case 5: // Execute instruction and return
result
       Teleduino328.instruction(data);
```

```
      Client.write('!');
      for(int i = 0; i < data[1] + 2; i++)
      {

Client.write(Teleduino328.hexEncode(data[i]
/ 16));

Client.write(Teleduino328.hexEncode(data[i]
% 16));
      }
      Client.write('\n');
      lastInstruction = millis();
      stage = 0;
      break;
    }
  }
  else
  {
    Teleduino328.setStatusLed(10);
    Teleduino328.reset();
  }

  // Has the instruction timeout been reached?
  if(millis() - lastInstruction > 30000)
  {
    Client.flush();
    Client.stop();
    Teleduino328.setStatusLed(9);
    Teleduino328.reset();
  }

  // Process refreshes every 50ms
  if(millis() - lastRefresh >= 50)
  {
    Teleduino328.pinTimers();
    Teleduino328.shiftRegisterTimers();
    Teleduino328.shiftRegisters();
    lastRefresh = millis();
  }

  // Check to see if reset has been requested
  Teleduino328.checkReset();
}
```

8.2 CODE FOR HAS APP: BLUETOOTH

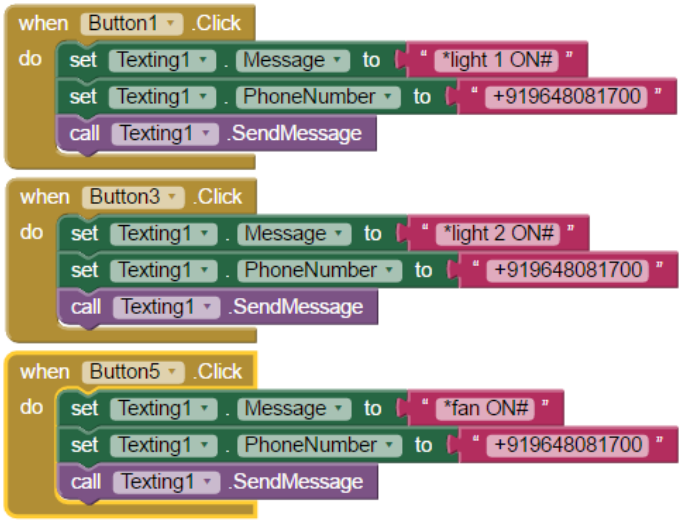
Each page of the app has a unique block of codes. The block for the Bluetooth screen is:

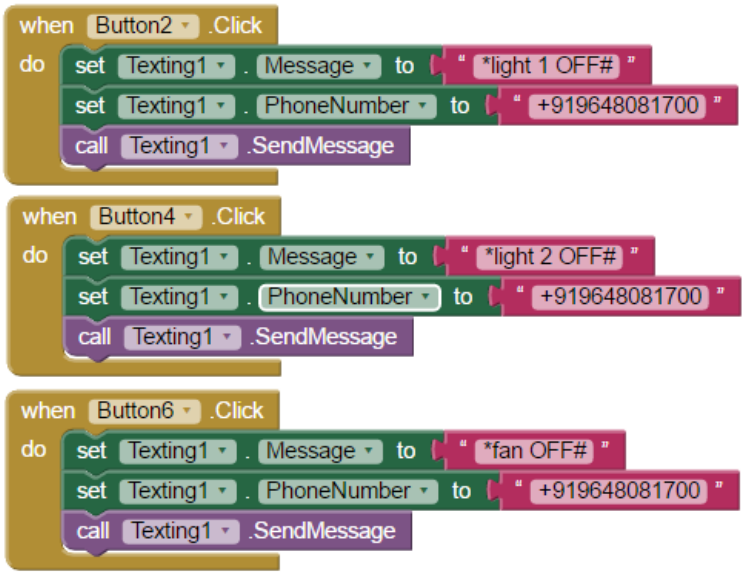


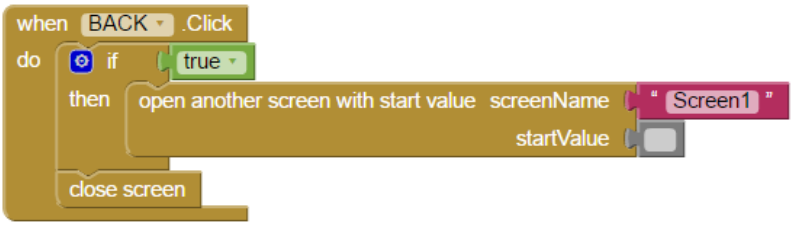


8.3 CODE FOR HAS APP: GSM

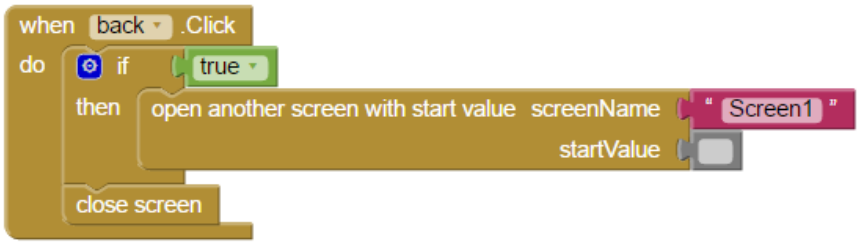
The blocks for the screen 2 GSM are:

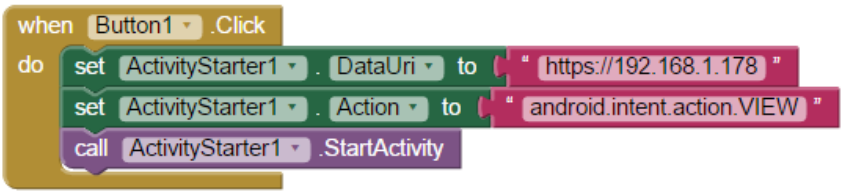






8.4 CODE FOR HAS APP: ETHERNET





CONCLUSION

In this project, a home automation system based on a Bluetooth wireless technology, GSM and Ethernet is proposed. The Bluetooth home network is designed for monitoring and remote control of different appliances connected over Bluetooth network in a home environment. The developed Bluetooth home network system includes emulation programs of each device and a home server program. The usefulness of the proposed method is proven through simulations and experiments using the developed Bluetooth device module.

Design and implementation of the GSM Home Automation System using the App Inventor for Android mobile phone has been discussed. The purpose of the GSM is to use mobile phone's inbuilt SMS facility and GSM Modem for automation of Home Appliances. Different hardware and software unit of the GSM Module is described. The complete application software has been designed using App Inventor for Android and C Language. The GSM application program is tested on various Android mobile phones which are satisfactory and responses received are encouraging. Similarly, the use of Ethernet Shield has enabled the user to control the appliances via the internet, remotely and successfully using the HAS app.

REFERENCES

[1] The official Bluetooth website from Bluetooth SIG www.Bluetooth.com, Date viewed: March 21, 2001, Bluetooth Specification Version 1.1.

[2] Bluetooth Committee, Specifications of the Bluetooth System (Core), December 1999, V1.0B.

[3] Home System Specification, EHS, 1997.

[4] Ericsson Mobile Communications AB, User Manual—Bluetooth PC Reference Stack, 1543 VNX 2/901 184 Uen Version R1a, April 2000.

[5] I.S. McKenzie, The 8051 Microcontroller, Prentice Hall, Upper Saddler River, NJ, 1999.

[6] Phillips Semiconductors, 87C51 8-bit Microcontroller Data Sheet, 1999.

[7] GSM Modems, http://www.nowsms.com/doc/configuring-smsc-connections/gsm-modems

[8] M. N. Jivani, Sharon Panth, "Home Appliance Control Ad-hoc Network System using App Inventor", International Journal of Emerging Technologies and Applications in Engineering, Technology and Sciences (IJ-ETA-ETS),

[9] USB Design By Example, A Practical Guide of Building I/O Devices, John Hyde, Wiley, New York, 1999.

[10] Philips Semiconductors, The I2C Bus Specification, Version 2.1, January 2000.

[11] Philips Semiconductors, The I2C Bus and How to Use It (Including Specification, Version 1.0), Technical Manual, 1997.