B.M.S COLLEGE OF ENGINEERING

Department of Electronics and Communication Engineering

Bull Temple Road, Basavanagudi, Bangalore - 560 019

Project Report

on

"Home Automation using LPC1343 and ESP8266"

Submitted by

Name of the Student	USN
CHANDAN G	1BM14EC019
CHINMAYI B TEGGI	1BM14EC022
DHEERAJ D KAMATH	1BM14EC027
JAHNAVI REDDY	1BM14EC042

Name of the Faculty In-charge: GEETHISHREE MISHRA

Designation: Assistant professor, ECE Department

CONTENTS

INTRODUCTION:

With the advent of Internet of Things, Automation is a growing area, ready to be used in everyday life.

Like every other advancement in household devices, the main aim is to increase the convenience of the user, while keeping power conservation in mind.

We want automation everywhere possible, allowing us to perform other tasks on hand. Be it a simple task of switching on the lights or doing the laundry. Automation allows the users to control devices from afar, and not exactly go up to manual switches, providing user comfort. It may also involve controlling the devices based on various parameters such as daylight, temperature etc.

Home automation or **smart home**(also known as **domotics**) is building automation for the home. It involves the control and automation of lighting, heating (such as smart thermostats), ventilation, air conditioning (HVAC), and security, as well as home appliances such as washer/dryers, ovens or refrigerators/freezers. Wi-Fi is often used for remote monitoring and control. Home devices, when remotely monitored and controlled via the Internet, are an important constituent of the Internet of Things.

Modern systems generally consist of switches and sensors connected to a central hub sometimes called a "gateway" from which the system is controlled with a user interface that is interacted either with a wall-mounted terminal, mobile phone software, tablet computer or a web interface, often but not always via Internet cloud services.

Early home automation began with labour-saving machines. Self-contained electric or gas-powered home appliances became viable in the 1900s with the introduction of electric power distribution and led to the introduction of washing machines (1904), water heaters (1889), refrigerators, sewing machines, dishwashers, and clothes dryers.

In the near future, home automation may be standardized to let us truly take advantage of all the possibilities. For the time being, the home security providers that specialize in home automation have focused on the most critical and useful parts of a connected home. At a basic level, this means the doors and windows and environmental devices (thermostat, smoke detectors, temperature, humidity, fire and carbon dioxide sensors) that keep us safe and comfortable. For additional real-time security, convenience, and control, home automation systems from security providers should also include options for video cameras. With the best systems, we'll also be able to include lights and individual electrical outlets into our home automation package.

PROBLEM STATEMENT:

Automation, as the name suggests, is all about reducing labor from the user. It aims at making tasks convenient and easier for the customer. Automation also aims at device control based on other parameters, so as to reduce unnecessary power consumption.

Here, we aim at controlling light based on three parameters:

1. Intentions of the user.
2. The presence of light.
3. The presence of the user.

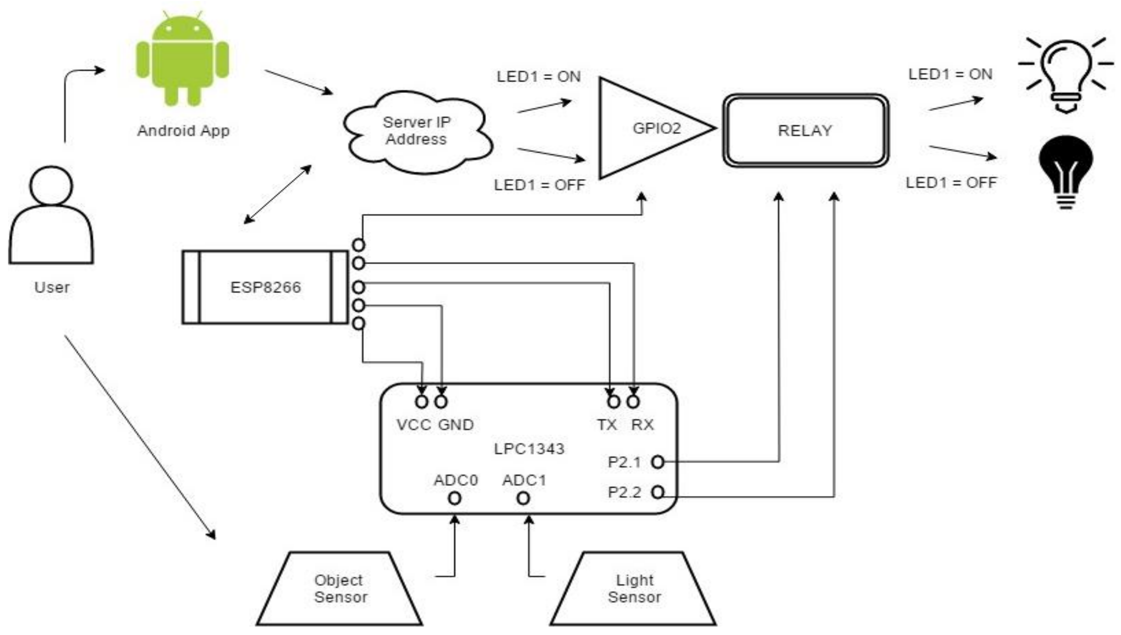
We require the light to be controlled by the user through an app. We use an ESP8266 module to create a server and help control the bulb when we receive commands from the app.

The system must also switch on the bulb when there is enough darkness. This is achieved by the light sensor, which uses an LDR.

The bulb must also turn off when the user leaves the space and forgets to turn off the device, consuming power. An object sensor can be used to detect the presence and control the bulb accordingly.

PROPOSED SOLUTION:

In order to achieve our goals, we implement a system as shown:



The main components include:

- ESP8266
- LPC1343
- Object Sensor
- Light Sensor
- 8-channel relay

The user can control the light through an app or let it automatically operate according to the outputs of the object and light sensors.

The ESP8266 module is a self-sufficient SOC module, that operates independently. We program the module to create a server that will respond to commands from the app and help control the bulb through the GPIO2 pin of the ESP8266 module.

The ESP module is powered from the LPC1343 board, using its 3.3V power supply. Note that the Tx and Rx pins of the module must still be enabled for it to work. This is done by simply using the Tx and Rx pins from the LPC1343 board.

When the ESP module is initialised, it connects to the given Wi-Fi and creates a server of its own with a unique IP address. The app must also be connected to the same Wi-Fi and the IP address.

When the server receives a command from the app, it interacts with the ESP8266 module to control the GPIO2 pin, which is connected to a relay. For example, when an ON command is received, the server sends the value to the module, which is programmed to make the GPIO2 pin go high when the ON command is received. This turns the relay circuit on, switching on the bulb. This resolves our aim to achieve user control.

For automatic control of the bulb based on either intensity of light or presence of a person, we use an object sensor and light sensor respectively. The sensors are interfaced to the LPC1343 board, through the ADC pins.

The analog outputs of the sensors are converted into digital values for processing using an Analog to Digital conversion program. These digital values are used to drive P2.2 and P2.2 pins on the boards that are defined as output pins here. These pins are again connected to the relay. It again operates in a similar way as described above.

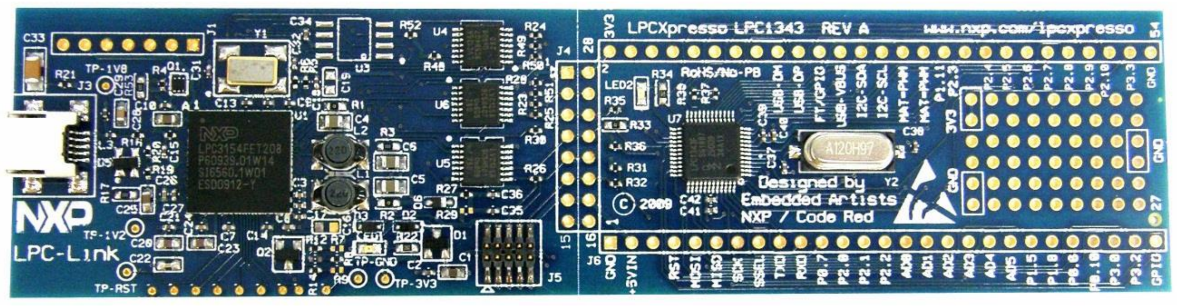
Hence we obtain control of the bulb in three different ways, achieving automation, power conservation, and user comfort.

The system can be further improved using the NodeMCU module, which is an advanced version of ESP8266. Multiple sensor outputs can be multiplexed and given to a single relay, reducing the hardware usage.

LPC1343

LPC is a family of 32-bit microcontroller integrated circuits by NXP Semiconductors (formerly Philips Semiconductors). The LPC chips are grouped into related series that are based on the same 32-bit ARM processor core, such as the Cortex-M4F, Cortex-M3, Cortex-M0+, or Cortex-M0.

The NXP LPC1300-series are based on the ARM Cortex-M3 core.



mxBaseBoard is the new addition to BlueBoard line from NGX Technologies. This board is intended to extend the functionality of the mx-LPC1343F board.

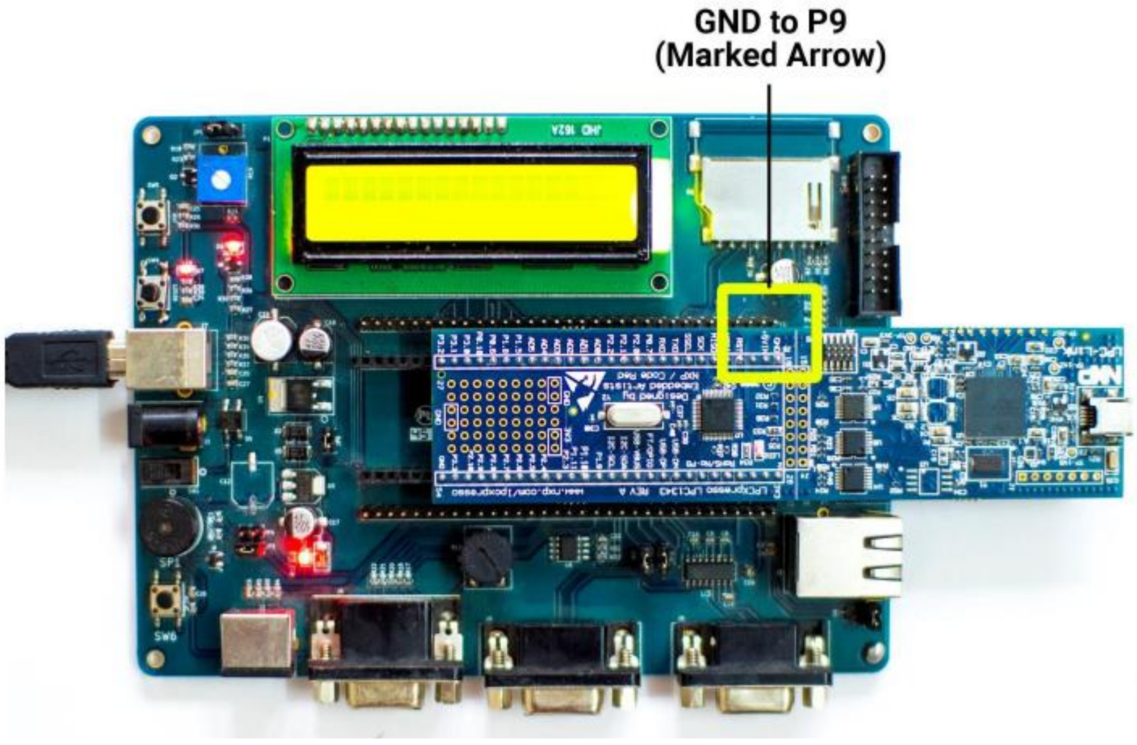
mX Base Board

Features include:

- 2x16 LCD with contrast control and backlight
- SD Card Connector
- USB
- Power Jack
- Power Switch
- Reset Button
- ISP Button
- External Interrupt Button
- Buzzer
- Audio Jack
- PS/2
- Serial Connector 0
- Preset for ADC
- On Board EEPROM

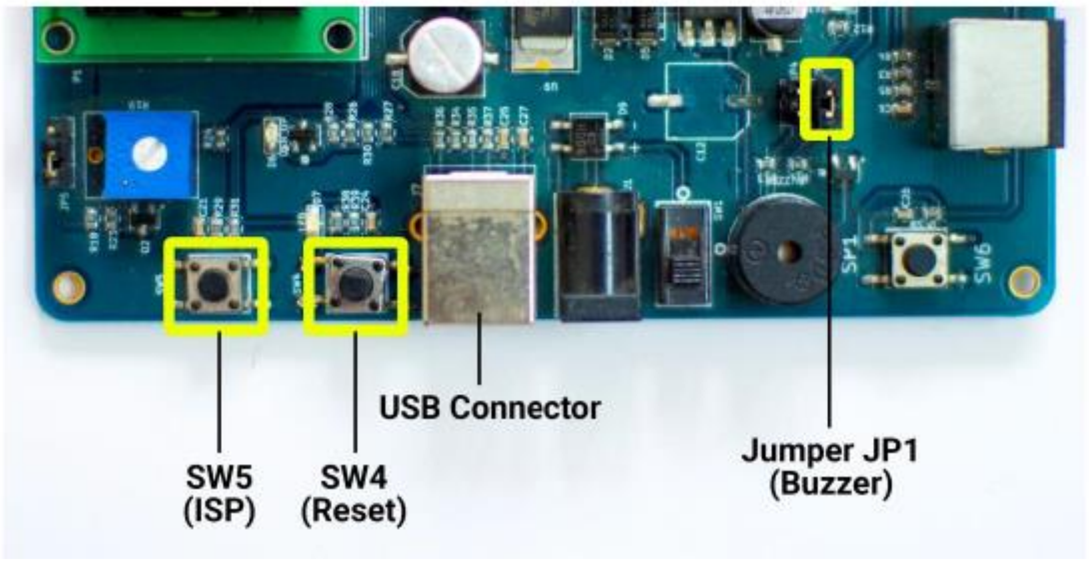
Mounting the mx-LPC1343F Board

The mx-LPC1343F Board must be mounted with a particular alignment. The GND pin on the mx-LPC1343F should be aligned with the pin 1 of the P9 female header on the BaseBoard. Refer to the marking as shown in the image below:

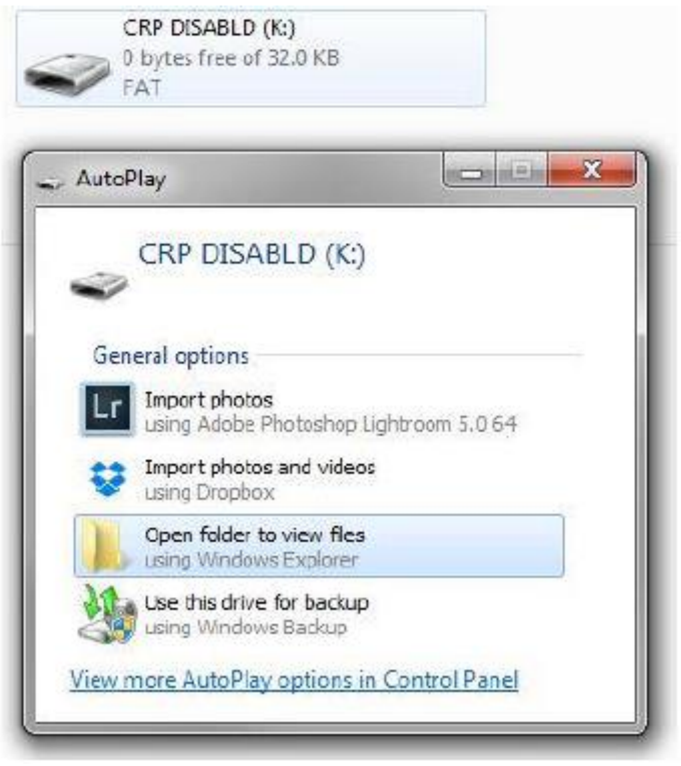


Programming the Board

To set the board into ISP mode, we need to press and hold SW5. Then press reset switch using SW4. Release the reset switch and SW4 and then release SW5.



The Device will be detected as external storage. Delete the existing Firmware.bin file and add your .bin file in place of it. Press reset and you have successfully uploaded the code onto the board.


CRP DISABLD (K:)
0 bytes free of 32.0 KB
FAT

AutoPlay

CRP DISABLD (K:)

General options

Import photos
using Adobe Photoshop Lightroom 5.0 64

Import photos and videos
using Dropbox

Open folder to view files
using Windows Explorer

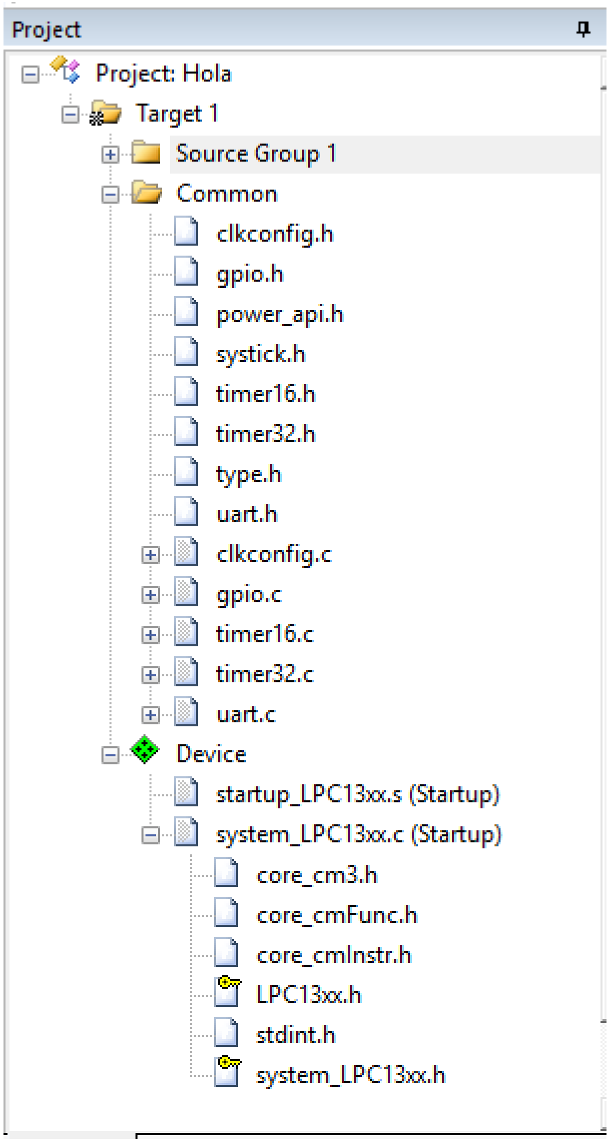
Use this drive for backup
using Windows Backup

View more AutoPlay options in Control Panel

Keil v5 IDE

Keil® MDK is the most comprehensive software development solution for ARM®-based microcontrollers and includes all components that you need to create, build, and debug embedded applications. Keil MDK Version 5 is the latest release of software development environment for a wide range of ARM Cortex-M based microcontroller devices.

Before we begin writing our C Program, we need to import different libraries which define the architecture and also let us use its varied functionalities.



```
Project
Project: Hola
  Target 1
    Source Group 1
    Common
      clkconfig.h
      gpio.h
      power_api.h
      systick.h
      timer16.h
      timer32.h
      type.h
      uart.h
      clkconfig.c
      gpio.c
      timer16.c
      timer32.c
      uart.c
    Device
      startup_LPC13xx.s (Startup)
      system_LPC13xx.c (Startup)
        core_cm3.h
        core_cmFunc.h
        core_cmInstr.h
        LPC13xx.h
        stdint.h
        system_LPC13xx.h
```

These header and c files initialize all the registers in the architecture and perform different functions. By just reading the function definition, we can implement the functions directly making it easier, without actually knowing what the internal working is.

Once all the required files have been added, write the following code:

```
#include "definitions.h"
#include "LPC13xx.h"
#include "gpio.h"
#include "libs/lcd.h"
#include "timer16.h"
#include "adc.h"
#include "uart.h"

#define PORT LED_PORT
#define PIN LED_PIN

extern volatile uint32_t ADCValue[ADC_NUM];
extern volatile uint32_t ADCIntDone;
extern volatile uint32_t OverRunCounter;

volatile uint32_t ADCMinValue[ADC_NUM];
volatile uint32_t ADCMaxValue[ADC_NUM];
volatile uint32_t ADCConversionCounter = 0;
volatile uint32_t ADCConversionBurstCounter = 0;

extern volatile uint32_t UARTCount;
extern volatile uint8_t UARTBuffer[BUFSIZE];
static int32_t ClearCounter = ADC_DEBUG_CLEAR_CNT;
static uint32_t DebugUpdateCounter = 0;
static uint32_t AnalogData;
static uint32_t i = 0;

uint8_t DigitalData;

static uint8_t ConvertDigital ( uint8_t digit )
{
  static uint8_t hex[] = "0123456789ABCDEF";
  return hex[digit & 0xf];
}

static void PrintoutADCValue( void )
{
  uint32_t i, j = 0;

  /* Bring cursor to home on terminal, write headers */
  UARTBuffer[j++] = (char)27;
```

```
UARTBuffer[j++] = '[';
UARTBuffer[j++] = 'H';
UARTBuffer[j++] = 'C';
UARTBuffer[j++] = 'U';
UARTBuffer[j++] = 'R';
UARTBuffer[j++] = ' ';
UARTBuffer[j++] = 'M';
UARTBuffer[j++] = 'I';
UARTBuffer[j++] = 'N';
UARTBuffer[j++] = ' ';
UARTBuffer[j++] = 'M';
UARTBuffer[j++] = 'A';
UARTBuffer[j++] = 'X';
UARTBuffer[j++] = '\r';
UARTBuffer[j++] = '\n';
UARTSend( (uint8_t *)UARTBuffer, j );

/* Update current, min, and max values on terminal */
for ( i = 0; i < ADC_NUM; i++ )
{
  j = 0;
  UARTBuffer[j++] = ConvertDigital( (uint8_t)((ADCValue[i]>>8)&0xF));
  UARTBuffer[j++] = ConvertDigital( (uint8_t)((ADCValue[i]>>4)&0xF));
  UARTBuffer[j++] = ConvertDigital( (uint8_t)(ADCValue[i]&0xF));
  UARTBuffer[j++] = ' ';
  UARTBuffer[j++] = ConvertDigital( (uint8_t)((ADCMinValue[i]>>8)&0xF));
  UARTBuffer[j++] = ConvertDigital( (uint8_t)((ADCMinValue[i]>>4)&0xF));
  UARTBuffer[j++] = ConvertDigital( (uint8_t)(ADCMinValue[i]&0xF));
  UARTBuffer[j++] = ' ';
  UARTBuffer[j++] = ConvertDigital( (uint8_t)((ADCMaxValue[i]>>8)&0xF));
  UARTBuffer[j++] = ConvertDigital( (uint8_t)((ADCMaxValue[i]>>4)&0xF));
  UARTBuffer[j++] = ConvertDigital( (uint8_t)(ADCMaxValue[i]&0xF));
  UARTBuffer[j++] = '\r';
  UARTBuffer[j++] = '\n';
  UARTSend( (uint8_t *)UARTBuffer, j );
}

/* Show number of samples until next clear */
UARTBuffer[j=0] = '\r';
UARTBuffer[j++] = '\n';
UARTBuffer[j++] = ConvertDigital( (uint8_t)((ClearCounter>>8)&0xF));
```

```
  UARTBuffer[j++] = ConvertDigital( (uint8_t)((ClearCounter>>4)&0xF));
  UARTBuffer[j++] = ConvertDigital( (uint8_t)(ClearCounter&0xF));
  UARTSend( (uint8_t *)UARTBuffer, j );

  return;
}

static void ClearStats(void)
{
  int i;
  for ( i = 0; i < ADC_NUM; i++ )
  {
    ADCMinValue[i] = 0x3ff;
    ADCMaxValue[i] = 0;
  }
}


void BuzzInit(){
   GPIOSetDir(BUZZER_PORT, BUZZER_PIN, OUTPUT);
}

void init_devices()
{
	GPIOInit();
	init_lcd();
	BuzzInit();
	ADCInit(ADC_CLK);

}

void _delay_ms(int delayms){
  for(int i=0; i < 2000000 * delayms; i++){};
}

/* LCD Print String */
void lcd_welcome(){
	lcd_putstring(0, "Welcome to");
	lcd_putstring(1, "Home Automation");
	//lcd_print(1, 1, 70, 2);
}
```

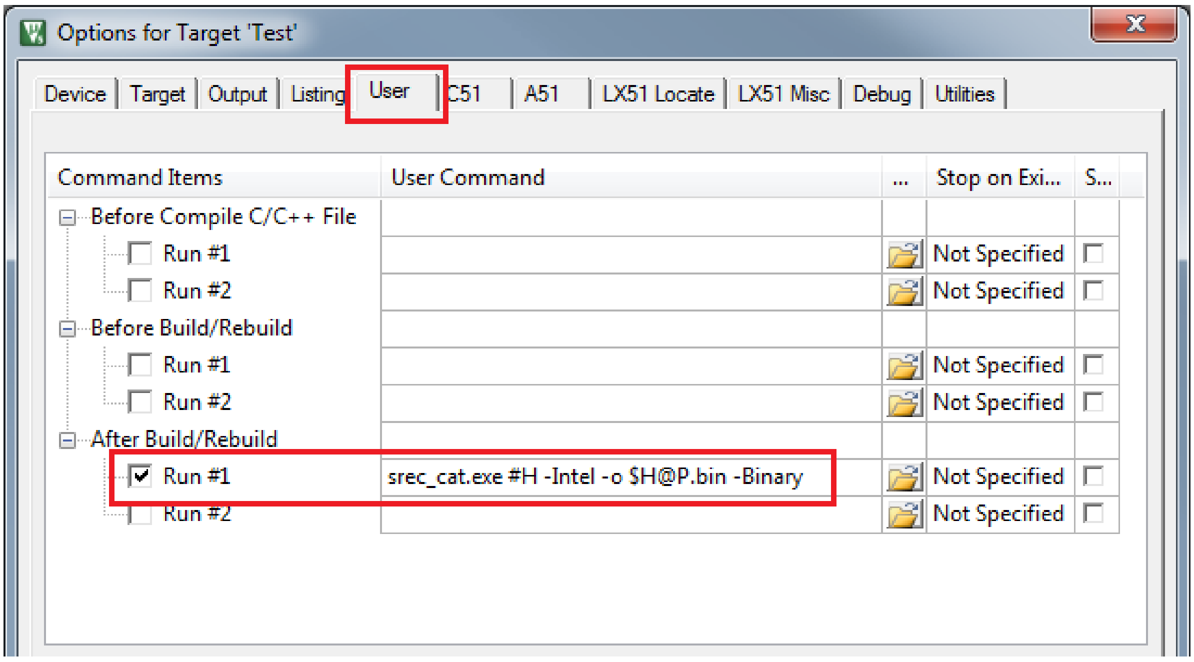
```
void BuzzLong(){
   GPIOSetValue(BUZZER_PORT, BUZZER_PIN, HIGH);
              _delay_ms(5);
              GPIOSetValue(BUZZER_PORT, BUZZER_PIN, LOW);
}

void LedBlink(){
               GPIOSetDir(PORT, PIN, OUTPUT);
    GPIOSetValue(PORT, PIN, HIGH);
               _delay_ms(2);
                          GPIOSetValue(PORT, PIN, LOW);
               _delay_ms(2);
}

static void UpdateStats(void)
{
  int i;
             for ( i = 0; i < ADC_NUM; i++ )
             {
             if ( ADCValue[i] < ADCMinValue[i])
                            ADCMinValue[i] = ADCValue[i];

               if ( ADCValue[i] > ADCMaxValue[i] )
                            ADCMaxValue[i] = ADCValue[i];

             }
}



void ADConversion(){
             float ValueVolts;

  for ( i = 0; i < ADC_NUM; i++ )
  {
      ADCValue[i] = ADCRead( i );          /* Polling */
                 //UpdateStats();
               }

                            DigitalData = ConvertDigital(ADCValue[0]);
```

```
                        ValueVolts = ADCValue[1];
                        lcd_clear();
                        lcd_putstring(0, "ADC Value");
                                  lcd_print(1, 1, ValueVolts, 5);
}

int main (void)
{
            /* Basic chip initialization is taken care of in SystemInit() called
              * from the startup code. SystemInit() and chip settings are defined
              * in the CMSIS system_<part family>.c file.
              */

  uint32_t i;
           init_devices();
           lcd_welcome();


#if BURST_MODE                                          /* Interrupt driven only */
  ADCBurstRead();
  while ( !ADCIntDone );
  ADCIntDone = 0;
#else
  for ( i = 0; i < ADC_NUM; i++ )
  {
#if ADC_INTERRUPT_FLAG                                     /* Interrupt driven */
           ADCRead( 5 );
           while ( !ADCIntDone );
           ADCIntDone = 0;
#else
           ADCValue[i] = ADCRead( i );    /* Polling */
#endif
                      #endif
                      lcd_print(1,1, ADCValue[0], 5);

  }

/*
int main()
{
           SystemCoreClockUpdate();
```

```
	init_devices();
	lcd_welcome();

	BuzzLong();


	while(1)
	{
		LedBlink();
		ADConversion();

	}
}
*/
```

Converting Hex to Bin

After this, select target options and under output tab, select the “generate hex file” and then rebuild the project. But since the LPC only accepts .bin files we will convert it in the following manner. In the User tab, add the following command as shown in the figure:



Command Items	User Command	...	Stop on Exi...	S...
Before Compile C/C++ File				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
Before Build/Rebuild				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
After Build/Rebuild				
☑ Run #1	srec_cat.exe #H -Intel -o $H@P.bin -Binary		Not Specified	☐
☐ Run #2			Not Specified	☐

Change the command to or add if blank: <$K\ARM\ARMCC\bin\fromelf.exe --bin --output=@L.bin !L>

Upload the .bin file so generated as shown previously.

THE ESP8266

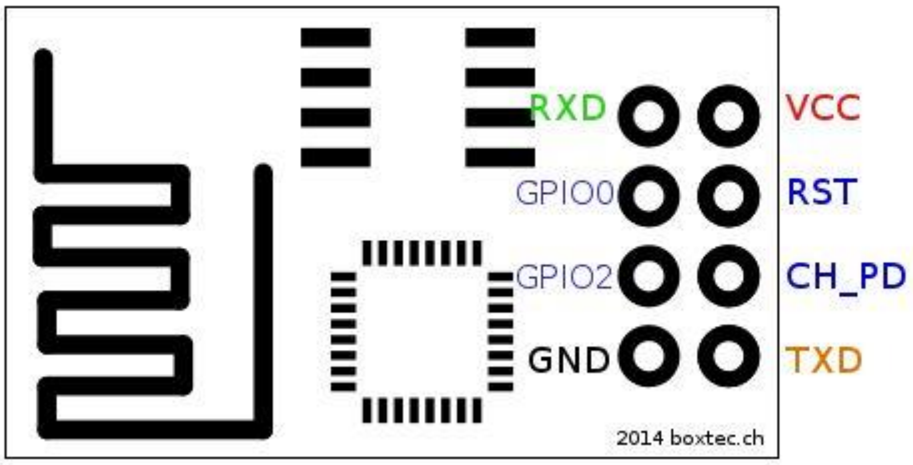
The ESP8266 Wi-Fi Module is a self-contained SOC with integrated TCP/IP protocol stack that can give any microcontroller access to your Wi-Fi network. The ESP8266 is capable of either hosting an application or offloading all Wi-Fi networking functions from another application processor. This small module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections.

Each ESP8266 module comes pre-programmed with an AT command set firmware, meaning, you can simply hook this up to your Arduino device and get about as much Wi-Fi-ability as a Wi-Fi Shield offers. The ESP8266 module is an extremely cost effective board with a huge, and ever growing, community.

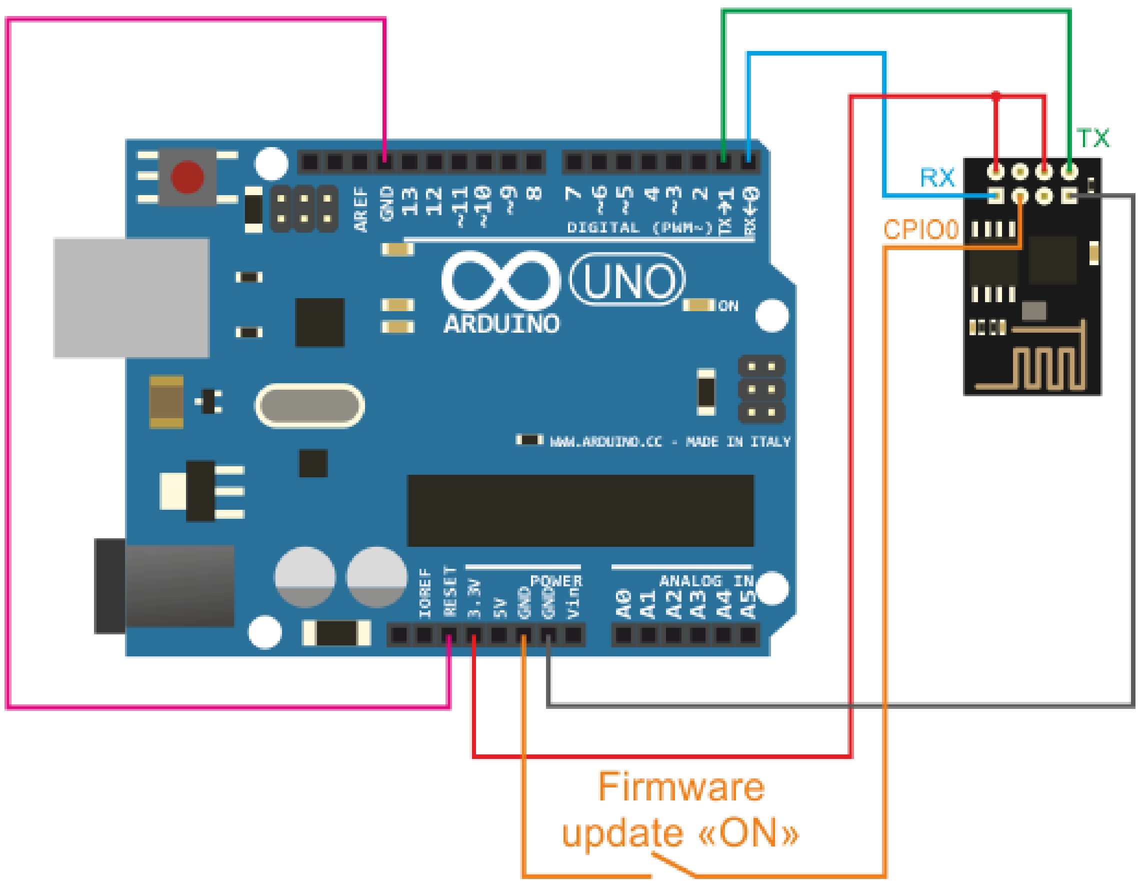
In simple words, the module creates a server, which can receive information from various sources. This information can be used to control different objects using the GPIO pins provided on the chip.

ESP8266 has 8 pins, 4 in the row of 2.

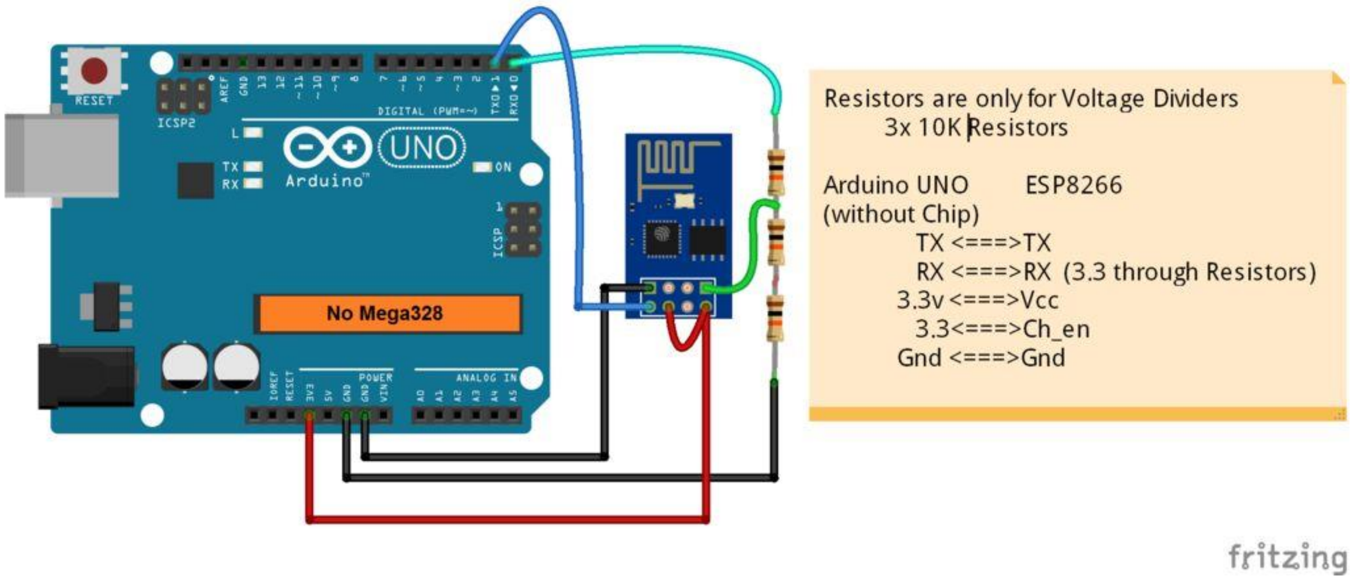
1. GND pin
2. VCC
3. GPIO 2 and 0 – General Purpose I/O pins.
4. RX and TX pins.
5. CH_PD (chip power-down) – an active low pin. Giving 3.3V to it will enable the chip.
6. RST(reset).



Interfacing ESP8266 with Arduino Uno



IMPORTANT NOTE: ESP8266 runs on 3.3V and its input pins are not 5V tolerant. To resolve this, we can use a voltage divider circuit. As it will be further established, we convert 5V from the LPC1343 board to 3.3V using a voltage divider consisting of 3 resistors having same resistance value as shown below:



Resistors are only for Voltage Dividers
3x 10K Resistors

Arduino UNO (without Chip)	ESP8266
TX <===>	TX
RX <===>	RX (3.3 through Resistors)
3.3v <===>	Vcc
3.3<===>	Ch_en
Gnd <===>	Gnd

Programming the ESP Module

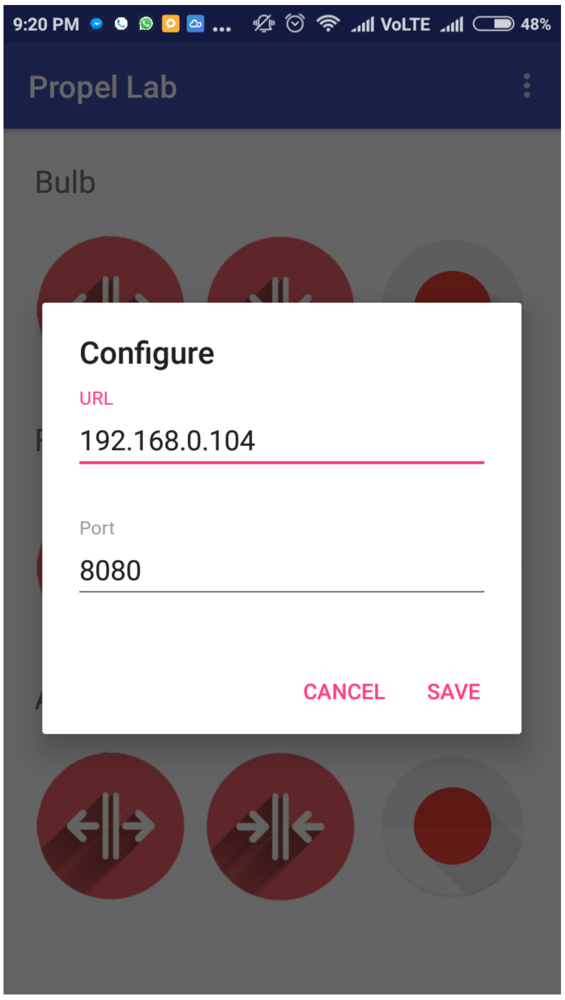
```
#include <ESP8266WiFi.h>

const char* ssid = "dk";
const char* password = "dheerajkamath";

int ledPin = 2; // GPIO13
WiFiServer server(80);

void setup() {
  Serial.begin(115200);
  delay(10);

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Connect to WiFi network
  Serial.println();
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");

  // Start the server
  server.begin();
  Serial.println("Server started");

  // Print the IP address
  Serial.print("Use this URL to connect: ");
  Serial.print("http://");
  Serial.print(WiFi.localIP());
  Serial.println("/");
```

```
}

void loop() {
  // Check if a client has connected
  WiFiClient client = server.available();
  if (!client) {
    return;
  }

  // Wait until the client sends some data
  Serial.println("new client");
  while(!client.available()){
    delay(1);
  }

  // Read the first line of the request
  String request = client.readStringUntil('\r');
  Serial.println(request);
  client.flush();

  // Match the request

  int value = LOW;
  if (request.indexOf("/LED1=ON") != -1)  {
    digitalWrite(ledPin, HIGH);
    value = HIGH;
  }
  if (request.indexOf("/LED1=OFF") != -1)  {
    digitalWrite(ledPin, LOW);
    value = LOW;
  }

// Set ledPin according to the request
digitalWrite(ledPin, value);

  // Return the response
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println(""); //  do not forget this one
  client.println("<!DOCTYPE HTML>");
  client.println("<html>");

  client.print("Led pin is now: ");

  if(value == HIGH) {
    client.print("On");
  } else {
    client.print("Off");
  }
  client.println("<br><br>");
  client.println("<a href=\"/LED=ON\"\"><button>Turn On </button></a>");
  client.println("<a href=\"/LED=OFF\"\"><button>Turn Off </button></a><br />");
  client.println("</html>");

  delay(1);
  Serial.println("Client disonnected");
  Serial.println("");

}
```

Once uploaded open the serial monitor and note down the IP address of the server so created by the module.

ANDROID APP

In Android Studio, create the layout as shown below:



9:20 PM ... VoLTE 48%

Propel Lab

Bulb

Configure

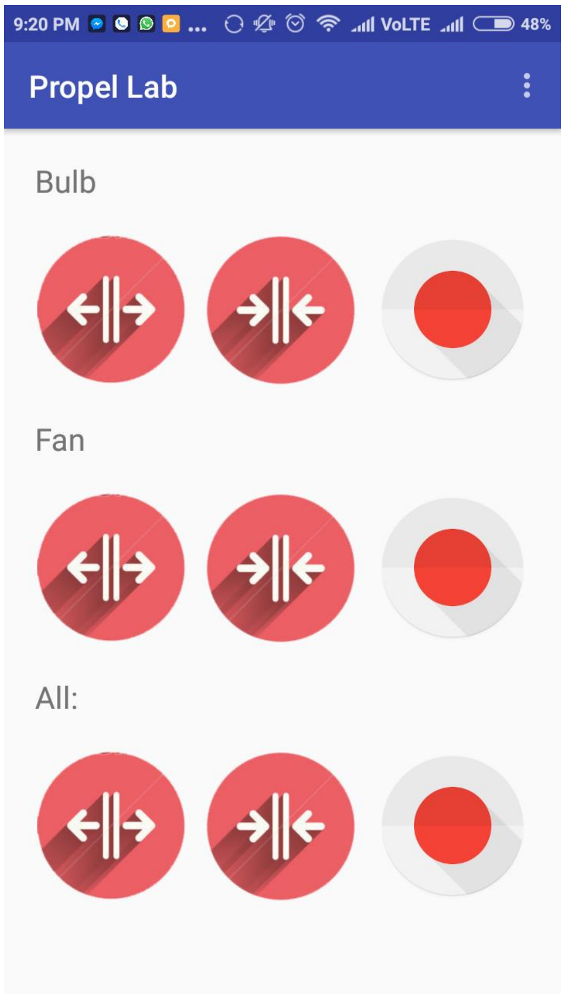
URL

192.168.0.104

Port

8080

CANCEL SAVE



9:20 PM ... VoLTE 48%

Propel Lab

Bulb

Fan

All:

You can download the Java Code from this link: http://www.github.com/Dheeraj22

In the settings, enter the details of the IP Address noted down earlier. When you click on first button in each row, it pings the following url: <IP Address>/LED1=ON and : <IP Address>/LED1=OFF for second button.

The ESP receives the command and when the string match occurs, GPIO2 becomes high or low. This output from the GPIO2 pin drives the relay to which the bulb is connected.

Since the output of ESP module is only 3.3V, it won't be possible to drive relays directly and hence we will use a 8 channel relay board which has logic level shifters which will convert the 3.3V to 5V enable.

POWER EFFICIENCY

Power efficiency can be improved by implementing an object sensor so that the ESP module remains ON only when the user is present.

The output of the Object Sensor is HIGH only when the User is present. This output is given to the LPC 1343 for ADC Conversion which is evident in the code in the ADCConversion() function. The function converts the analog to digital voltage. This digital voltage is given to CH_PD of the ESP module which is called the enable pin.

ISSUES IN AUTOMATION

Though automation can be a great way to seek comfort and make tasks easier, it poses a number of issues:

- A WiFi network connected to the internet can be vulnerable to hacking. Therefore, security must be a major parameter to be considered when we install such systems.
- Technology is still in its infancy, and consumers could invest in a system that becomes abandonware. In 2014, Google bought the company selling the Revolv Hub home automation system, integrated it with Nest and in 2016 shut down the servers Revolv Hub depended on, rendering the hardware useless.
- automation is at an intermediate level of intelligence, powerful enough to take over control that used to be done by people, but not powerful enough to handle all abnormalities. Moreover, its level of intelligence is insufficient to provide the continual, appropriate feedback that occurs naturally among human operators. This is the source of the current difficulties.
- Microsoft Research found in 2011, that home automation could involve high cost of ownership, inflexibility of interconnected devices, and poor manageability
- Automation is also leading a steady decrease in job opportunities in some fields, though is an emerging field for electronics enthusiasts.

CONCLUSION:

Thus Home Automation is a vast area which offers room for improvements and more devices can be connected to this system by using the advanced version of ESP8266 called the NodeMCU which has 13GPIO pins. These pins can be used to drive the 8 relays.

Further areas of research can include safety, comfort and conservation.

EVALUATION SHEET:

Sl No.	Name of the Student	USN	Topic & Report 3 Marks	Presentation/Viva 2 Marks	Total 5 Marks
1					
2					
3					
4					

Signature of the Faculty In charge