A Mini Project Report on

**Home Automation Using Google Assistant**

Submitted to the

****

Savitribai Phule Pune University

In partial fulfillment for the award of the

Degree Bachelor of Engineering In

Computer Engineering

By

**Kartik Milind Bhavsar(45)**

**Yash Anil Jadhav(24)**

**Patil Aayusha Sandip(47)**

**Bhadane Shreejta Nandkishor(22)**

**Ankita Kailas Aher(39)**

**SNJB’s Late Sau. Kantabai Bhavarlalji Jain, College of**

**Engineering, Chandwad Dist: Nashik**

**1. INTRODUCTION**

Today in the headway of Automation innovation, life is getting simpler and less demanding in all spheres. Home automation is a modern technology that modifies your home to perform different sets of task automatically. No wonders, home automation in India is already the buzz word, especially as the wave of second generation home owners grows, they want more than shelter, water, and electricity. There is no shortage of possibilities for smart home IoT devices as home automation seems to be the wave of the future. The requirement for Office and Home automation arises due to the advent of IoT, in a big way in homes and office space. These smart automation devices happen to have an interface with IoT. IT automation will be the key to bridging the gap between human limitations and technology’s capabilities. The modern homes are automated through the internet and the home appliances are controlled. The user commands over the internet will be obtained by the Wi-Fi modems. The Microcontroller has an interface with this modem. The system status is displayed through the LCD display, along with the system data. This is a typical IoT based Home Automation system, for controlling all your home appliances.

This paper proposes such system that uses the Google Assistant, the IFTTT (if this than that) application, the Blynk application and the NodeMCU which the main microcontroller and used as major components along with a relay module. Human voice is used to give commands to the Google Assistant. All these components are connected over the internet using Wi-Fi which puts this system under the Internet of Things (IOT).

# 2. PROPOSED SYSTEM

In this project, nodeMCU ESP8266 is put in with a blynk application account with the IFTTT application then added to google assistant for voice command. voice command control the home appliances. The proposed system eliminates the complication of wiring in case of wired automation. System’s operating range is more than the Bluetooth. The existing system doesn’t allow the monitoring and controlling of appliances via remote but in the proposed system that using the Wi-Fi based home automation system it allows to monitor and control the appliances. The block diagram of google assisatant based home automation shown below:

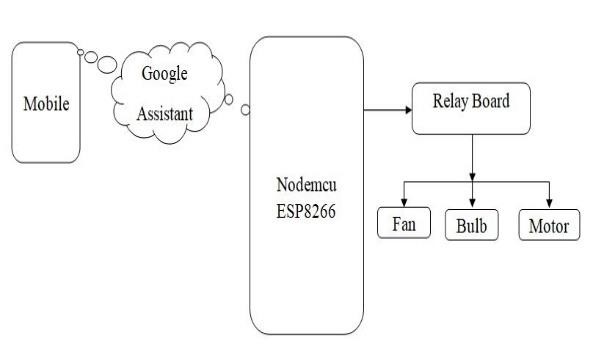


Fig 1: Block diagram of Proposed System

*2.1 Explanation of block diagram is as follow* In the block diagram shown in Fig:1, NodeMCU is linked with the Blynk application account with the IFTTT(if this than that)application which is connected to the Google assistance cloud. Home appliances bulb, fan, motor etc. are connected to the nodeMCU. The system consists of two main categories:

1. **Hardware:** It is the main control unit which has the ability to control the home appliances like bulb fan etc.

1. **Software:** Arduino IDE, Google Assistant.

## 2.1.1 Hardware component description

1. NodeMCU**:** NodeMCU is an open source lua based firmware and development board specially targeted for IOT badsed applications. It includes firmware that runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which is based on the ESP-12 module. The NodeMCU ESP8266 development board comes with the ESP-12E module containing ESP8266 chip having Tensilica Xtensa 32-bit LX106 RISC microprocessor. This microprocessor supports RTOS and operates at 80MHz to 160 MHz adjustable clock frequency. NodeMCU has 128 KB RAM and 4MB of Flash memory to store data and programs. Its high processing power with in-built Wi-Fi / Bluetooth and Deep Sleep Operating features make it ideal for IoT projects. NodeMCU can be powered using Micro USB jack and VIN pin (External Supply Pin). It supports UART, SPI, and I2C interface.

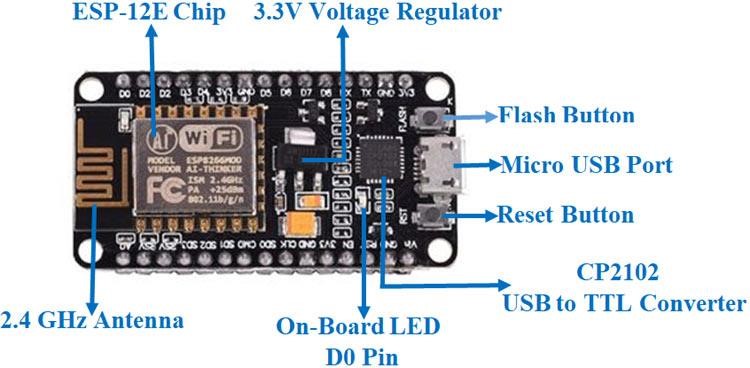


Fig 2: NodeMCU (ESP8266) Board

The NodeMCU Development Board can be easily programmed with Arduino IDE since it is easy to use. Programming NodeMCU with the Arduino IDE will hardly take 5-10 minutes. All we need is the Arduino IDE, a USB cable and the NodeMCU board itself.

1. **Relay:** The relay is the device that opens or closes the contacts to cause the operation of the other electric control. It detects the undesirable condition with an assigned area and gives the commands to the circuit breaker to disconnect the affected area through ON or OFF. Every electromechanical relay consists of

1. Electromagnet

3. Mechanically movable contact

1. Switching points and
2. Spring

There are 3 pins in the relay module, these are as follow:

COM: common pin

NO: normally open – there is any contact between the common pin and the normally open pin. So, when you trigger the relay, it connects to the COM pin and power is provided to the load.

NC: Normally closed – there is contact between the common pin and the normally closed pin. There is always connection between the COM and NC pins, even when the relay is turned off. When you trigger the relay, the circuit is opened and there is no supply provided to the load.

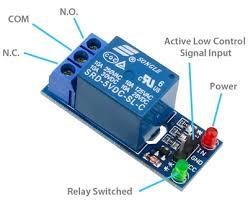


Fig 3: one channel relay module

It works on the principle of an electromagnetic attraction. When the circuit of the relay senses the fault current, it energizes the electromagnetic field which produces the temporary magnetic field. This magnetic field moves the relay armature for opening or closing the connections. The small power relay has only one contact, and the high power relay has two contacts for opening the switch.

**2.1.2 Software components description**

1. **Arduino IDE:** Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing. The Arduino software is easy-touse for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:
2. **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 Arduino modules cost less than $50.
3. **Cross-platform -** The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
4. **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.
5. **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, you can add AVR-C code directly into your Arduino programs if you want to.

Code : -

#ifdef ENABLE\_DEBUG

#define DEBUG\_ESP\_PORT Serial

#define NODEBUG\_WEBSOCKETS

#define NDEBUG

#endif

#include <Arduino.h>

#include <ESP8266WiFi.h>

#include "SinricPro.h"

#include "SinricProSwitch.h"

#include <map>

#define WIFI\_SSID "Redmi note 9"

#define WIFI\_PASS "kartik13"

#define APP\_KEY "10f5effd-f480-49e2-9996-89b4c6e1d77e"

#define APP\_SECRET "052ff997-5a79-46f6-9c18-d12ab2733dbd-fee44e5f-4096-4fc9-927a-16ed10a951a7"

//Enter the device IDs here

#define device\_ID\_1 "638352d6b8a7fefbd64b5b60"

#define device\_ID\_2 "6383531a333d12dd2affc0c1"

#define device\_ID\_3 "638353c4b8a7fefbd64b5c57"

#define device\_ID\_4 "K"

// define the GPIO connected with Relays and switches

#define RelayPin1 3 //D1

#define RelayPin2 1 //D2

#define RelayPin3 16 //D5

#define RelayPin4 5 //D6

#define SwitchPin1 4 //SD3

#define SwitchPin2 14 //D3

#define SwitchPin3 12 //D7

#define SwitchPin4 13 //RX

#define wifiLed 12 //D0

// comment the following line if you use a toggle switches instead of tactile buttons

//#define TACTILE\_BUTTON 1

#define BAUD\_RATE 9600

#define DEBOUNCE\_TIME 250

typedef struct { // struct for the std::map below

int relayPIN;

int flipSwitchPIN;

} deviceConfig\_t;

std::map<String, deviceConfig\_t> devices = {

//{deviceId, {relayPIN, flipSwitchPIN}}

{device\_ID\_1, { RelayPin1, SwitchPin1 }},

{device\_ID\_2, { RelayPin2, SwitchPin2 }},

{device\_ID\_3, { RelayPin3, SwitchPin3 }},

{device\_ID\_4, { RelayPin4, SwitchPin4 }}

};

typedef struct { // struct for the std::map below

String deviceId;

bool lastFlipSwitchState;

unsigned long lastFlipSwitchChange;

} flipSwitchConfig\_t;

std::map<int, flipSwitchConfig\_t> flipSwitches;

void setupRelays() {

for (auto &device : devices) {

int relayPIN = device.second.relayPIN;

pinMode(relayPIN, OUTPUT);

digitalWrite(relayPIN, HIGH);

}

}

void setupFlipSwitches() {

for (auto &device : devices) {

flipSwitchConfig\_t flipSwitchConfig;

flipSwitchConfig.deviceId = device.first;

flipSwitchConfig.lastFlipSwitchChange = 0;

flipSwitchConfig.lastFlipSwitchState = true;

int flipSwitchPIN = device.second.flipSwitchPIN;

flipSwitches[flipSwitchPIN] = flipSwitchConfig;

pinMode(flipSwitchPIN, INPUT\_PULLUP);

}

}

bool onPowerState(String deviceId, bool &state)

{

Serial.printf("%s: %s\r\n", deviceId.c\_str(), state ? "on" : "off");

int relayPIN = devices[deviceId].relayPIN;

digitalWrite(relayPIN, !state);

return true;

}

void handleFlipSwitches() {

unsigned long actualMillis = millis();

for (auto &flipSwitch : flipSwitches) {

unsigned long lastFlipSwitchChange = flipSwitch.second.lastFlipSwitchChange;

if (actualMillis - lastFlipSwitchChange > DEBOUNCE\_TIME) {

int flipSwitchPIN = flipSwitch.first;

bool lastFlipSwitchState = flipSwitch.second.lastFlipSwitchState;

bool flipSwitchState = digitalRead(flipSwitchPIN);

if (flipSwitchState != lastFlipSwitchState)

#ifdef TACTILE\_BUTTON

if (flipSwitchState) {

flipSwitch.second.lastFlipSwitchChange = actualMillis;

String deviceId = flipSwitch.second.deviceId;

int relayPIN = devices[deviceId].relayPIN;

bool newRelayState = !digitalRead(relayPIN);

digitalWrite(relayPIN, newRelayState

SinricProSwitch &mySwitch = SinricPro[deviceId];

mySwitch.sendPowerStateEvent(!newRelayState);

}

#endif

flipSwitch.second.lastFlipSwitchState = flipSwitchState;

}

}

}

}

void setupWiFi()

{

Serial.printf("\r\n[Wifi]: Connecting");

WiFi.begin(WIFI\_SSID, WIFI\_PASS);

while (WiFi.status() != WL\_CONNECTED)

{

Serial.printf(".");

delay(250);

}

digitalWrite(wifiLed, LOW);

Serial.printf("connected!\r\n[WiFi]: IP-Address is %s\r\n", WiFi.localIP().toString().c\_str());

}

void setupSinricPro()

{

for (auto &device : devices)

{

const char \*deviceId = device.first.c\_str();

SinricProSwitch &mySwitch = SinricPro[deviceId];

mySwitch.onPowerState(onPowerState);

}

SinricPro.begin(APP\_KEY, APP\_SECRET);

SinricPro.restoreDeviceStates(true);

}

void setup()

{

Serial.begin(BAUD\_RATE);

pinMode(wifiLed, OUTPUT);

digitalWrite(wifiLed, HIGH);

setupRelays();

setupFlipSwitches();

setupWiFi();

setupSinricPro();

}

void loop()

{

SinricPro.handle();

handleFlipSwitches();

}

OUTPUT : -

