IoT Based Home Automation System

Table of Contents

ABSTRACT

Abstract

The development of an intelligent smart home automation system using ESP32 as the core microcontroller represents a significant advancement in modern home technology. This system integrates Home Assistant and Alexa for remote monitoring and voice-controlled operation of various home appliances, significantly enhancing user comfort and convenience.

At its core, the project leverages ESP32's capabilities in data processing and wireless communication to establish a robust network connecting the smart home automation system with Home Assistant and Alexa platforms. This enables seamless integration and mutual communication between these components, facilitating sophisticated automation rules and workflows.

To ensure accurate environmental monitoring and dynamic device control, the system incorporates DHT11 sensors, which measure temperature and humidity levels. These sensor readings are crucial for adjusting systems such as air conditioners in real-time based on room conditions, thereby optimizing energy usage and comfort.

The 4-channel relay module is a pivotal component of this project, allowing the control of up to four different electrical devices through ESP32. This feature enhances the system's versatility and capability to manage multiple tasks simultaneously. The setup includes a 12V power supply with a buck converter to ensure stable and reliable power distribution from the source to ESP32 and other components.

The Home Assistant OS acts as the management layer, orchestrating complex automation rules and integrating with Alexa for voice-controlled commands. This dual-layered integration allows users to interact with their home devices through both traditional remote interfaces and voice commands, providing an intuitive user experience.

One of the key advantages of this project is its ability to enhance convenience by enabling remote monitoring and control of home devices. Additionally, it promotes energy efficiency through optimized temperature-based air conditioning adjustments. The system also offers enhanced

security by ensuring control access even when away from the home, and it provides scalability for future integrations with additional sensors and devices.

Cost-effectiveness is another significant benefit, as the use of ESP32, a widely available microcontroller at an affordable price point, makes this project budget-friendly without compromising on functionality. The system's modular design allows for potential upgrades or expansions in the future, ensuring adaptability to evolving user needs.

Introduction

1.1 Home Automation Overview

Home automation is the integration of technology into household systems to provide automated control over appliances, lighting, heating, cooling, security, and entertainment devices. This concept aims to enhance convenience, improve energy efficiency, and increase security by allowing users to monitor and manage devices remotely.

In the past, home automation was primarily limited to high-end, proprietary systems that required professional installation and significant investment. However, with advancements in IoT (Internet of Things), microcontrollers, and wireless communication, smart home solutions have become more accessible, affordable, and easier to implement.

Modern home automation systems leverage various communication protocols like Wi-Fi, Zigbee, Z-Wave, and Bluetooth to establish a seamless connection between smart devices. This enables remote access, automation, and intelligent decision-making based on real-time data.

1.1.1 Key benefits of home automation include:

- **Convenience** – Users can control appliances with a mobile app, voice commands, or automated schedules.
- **Energy Efficiency** – Automated control of appliances reduces unnecessary power consumption.
- **Security** – Surveillance cameras, motion detectors, and smart locks enhance home security.
- **Cost Savings** – Optimized energy use leads to lower electricity bills.
- **Customization** – Personalized automation routines based on user preferences.

This project focuses on IoT-based home automation using the ESP32 microcontroller, which allows smart control of home appliances via Home Assistant and Alexa.

1.2 Role of IoT in Home Automation

The **Internet of Things (IoT)** is the backbone of modern home automation. IoT refers to a network of **connected smart devices that can communicate and exchange data** over the internet. These devices can operate autonomously or be controlled remotely, making them ideal for home automation.

1.2.1 Key components of an IoT-based home automation system include:

- **Microcontrollers** – Such as **ESP32, Raspberry Pi, or Arduino**, which process data and control appliances.
- **Sensors** – Temperature, motion, humidity, and light sensors collect environmental data.
- **Actuators** – Relays, smart switches, and motors execute control commands.
- **Cloud Platforms** – Enable remote access and data storage for smart devices.
- **Mobile Applications** – Provide an interface for users to monitor and control devices.

1.2.2 The ESP32 microcontroller **used in this project is ideal for IoT applications due to its:**

- **Built-in Wi-Fi & Bluetooth**, allowing seamless communication with smart home platforms.
- **Low power consumption**, making it energy-efficient.
- **High processing power**, enabling real-time automation.
- **Multiple GPIO pins**, supporting various sensors and actuators.

By integrating ESP32 with **Home Assistant and Alexa**, this project provides a **comprehensive home automation solution** that can be controlled via voice commands, mobile apps, and automated triggers.

1.3 Project Objective

This project aims to design and implement an **ESP32-based smart home automation system** with the following objectives:

1. **Automated Temperature-Based AC Control**
    - The system will use a **DHT11 temperature and humidity sensor** to monitor room conditions.
    - If the temperature exceeds a predefined threshold (e.g., 35°C), the **AC relay will automatically turn ON**.
    - If the temperature drops below a set limit (e.g., 30°C), the **AC relay will turn OFF**.
2. **Remote Control of Appliances via Home Assistant**
    - The ESP32 will be integrated with **Home Assistant**, allowing users to control appliances via a **smartphone or web dashboard**.
    - The system will provide real-time monitoring of device status.
3. **Voice Control with Alexa Integration**
    - The automation system will be compatible with **Amazon Alexa**, allowing users to control appliances using voice commands.
    - Example: Saying *"Alexa, turn on the bedroom light"* will activate the relay controlling the bedroom light.
4. **Wi-Fi Connectivity for IoT Integration**
    - The ESP32 will connect to a **Wi-Fi network (mobile hotspot or router)** to enable remote access.
    - Users can control appliances even when they are **away from home**.
5. **Energy Efficiency & Smart Scheduling**
    - The system will **reduce electricity consumption** by ensuring that appliances are turned OFF when not in use.
    - Scheduled automation routines will be implemented (e.g., turning lights ON at sunset).

By achieving these objectives, this project will provide a **low-cost, reliable, and efficient home automation system** using ESP32, Home Assistant, and Alexa.

1.4 Scope of the Project

The project covers various aspects of IoT-based home automation, focusing on:

1. Hardware Implementation

- Selection and configuration of **ESP32 microcontroller** as the central processing unit.
- Integration of sensors (**DHT11**) and actuators (**4-channel relay module**) for automation.
- Power management using a **12V power supply** and a **buck converter** to step down voltage for ESP32.

2. Software & Firmware Development

- Programming ESP32 using **ESPHome firmware** for easy integration with Home Assistant.
- Developing automation scripts in Home Assistant for **temperature-based control, remote access, and voice integration**.
- Securing **wireless communication** between ESP32 and the IoT platform**.**

3. User Interface & Control Mechanisms

- Configuring the **Home Assistant dashboard** for device monitoring and control.
- Integrating **Alexa** for **voice-activated control** of appliances.
- Enabling **manual control via smartphone** or a **web-based UI**.

4. Security & Data Protection

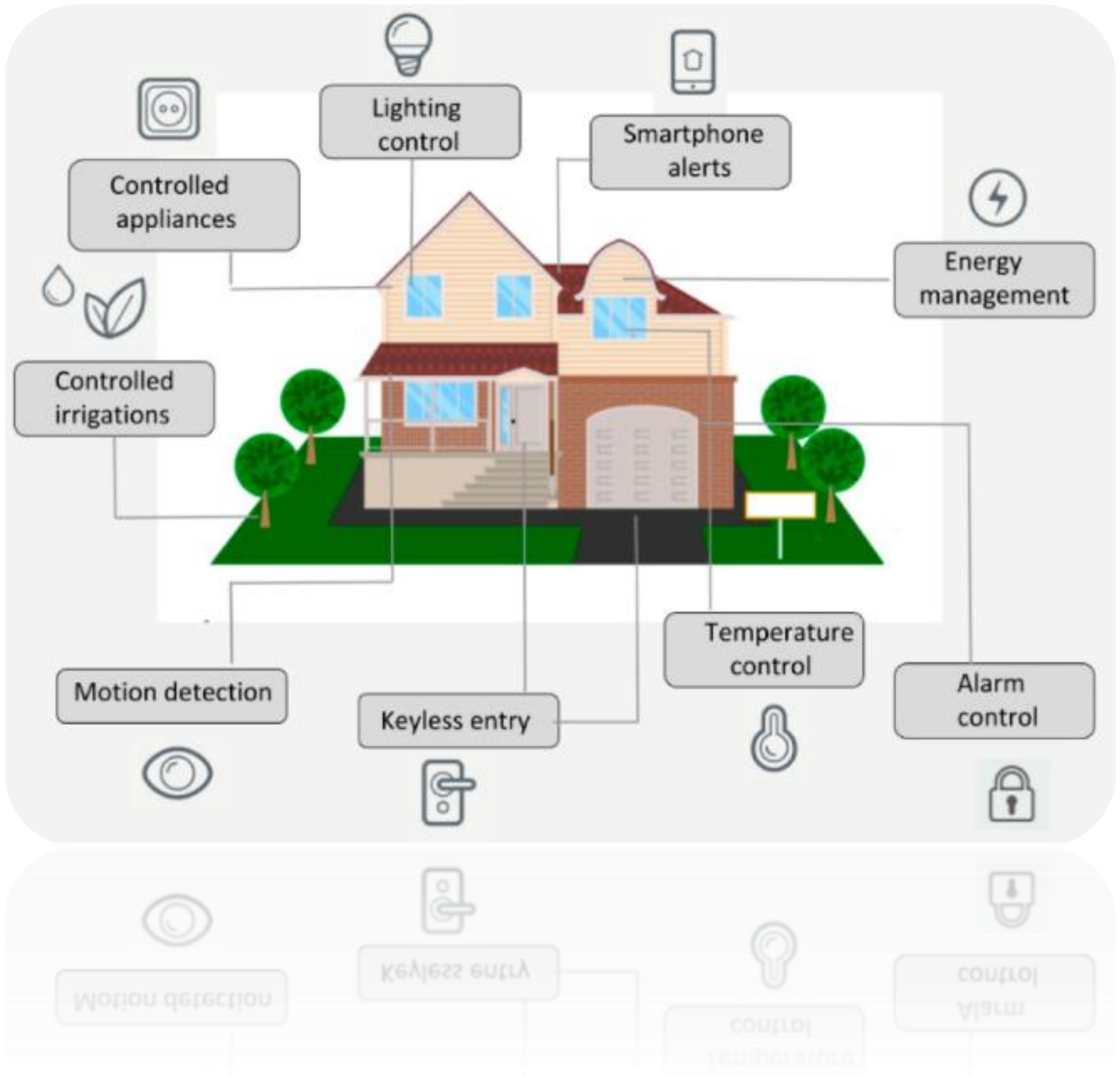
- Implementing **encrypted communication** between ESP32 and Home Assistant to prevent unauthorized access.
- Ensuring **reliable connectivity** and **failover mechanisms** for network issues.

5. Future Scalability & Expandability

The system is **designed for scalability**, allowing future enhancements such as:

- **Integration of motion sensors** for automated lighting control.
- **Smart power monitoring** to track and optimize energy usage.
- **Security camera integration** for real-time surveillance.
- **AI-based automation** for predictive control based on user behavior.

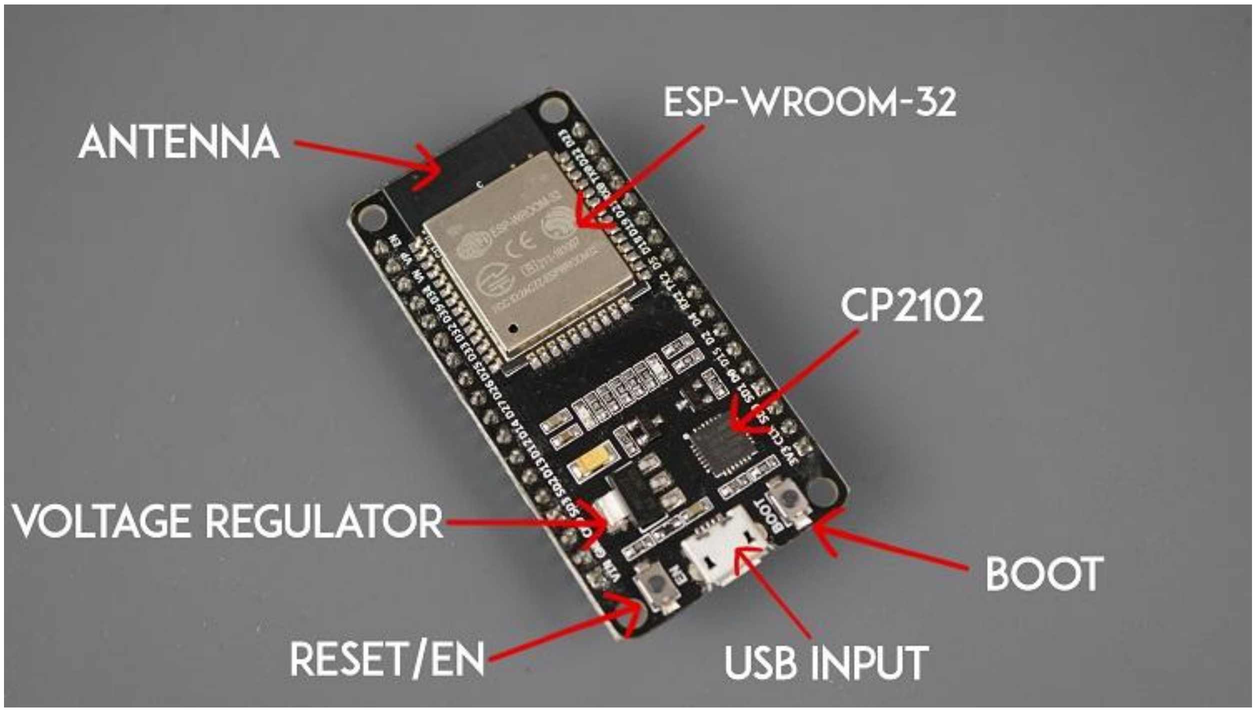
COMPONENTS USED



2.1 ESP32

ESP32 is a low-cost, power-efficient, and highly integrated microcontroller **designed for IoT applications. It features** built-in Wi-Fi and Bluetooth **capabilities, making it an ideal choice for smart home automation. Developed by** Espressif Systems**, the ESP32 is an** advanced version of the ESP8266**, offering** dual-core processing, more GPIO pins, and enhanced security features**.**

2.1.1 Key Features of ESP32



1. **Processor and Architecture**
    - Dual-core 32-bit Xtensa LX6 CPU (operating at 160–240 MHz).
    - Low power consumption with multiple sleep modes.
    - Supports real-time operating systems (RTOS) for multitasking.
2. **Connectivity**
    - Integrated Wi-Fi (802.11 b/g/n) for wireless communication.
    - Bluetooth 4.2 & Bluetooth Low Energy (BLE) for device pairing.
3. **I/O and Interfaces**

   - 34 General Purpose Input/Output (GPIO) pins for sensor and relay control.
   - Supports I2C, SPI, UART, ADC, DAC, PWM, and I2S communication.
4. **Memory & Storage**
   - Up to 520 KB SRAM and 4MB Flash Memory.
   - Supports external SD cards for additional storage.
5. **Security Features**
   - Encryption support (AES, RSA, SHA-2) for secure data transmission.
   - Secure Boot to prevent unauthorized firmware updates.

2.1.2 Why ESP32 for Home Automation?

ESP32 is widely used in home automation because of the following reasons:

Wireless Connectivity: Built-in Wi-Fi and Bluetooth allow it to communicate with smart home platforms like Home Assistant and Alexa.

Multiple GPIOs: Can control multiple relays, sensors, and displays for automation.

Energy Efficiency: Features deep sleep modes, reducing power consumption when idle.

Easy Integration: Supports ESPHome, MQTT, and HTTP protocols for seamless communication with IoT devices.

OTA Updates: Allows Over-the-Air firmware updates, reducing the need for manual reprogramming.

Affordable and Scalable: ESP32 is a low-cost microcontroller with high performance, making it perfect for large-scale smart home deployments.

2.1.3 ESP32 Pin Configuration

The ESP32 has multiple pins for various functions:

Pin Type	Function
GPIO Pins	Digital Input/Output for connecting relays, sensors, LEDs.
ADC (Analog to Digital Converter)	Reads analog signals from sensors (e.g., temperature, light).
DAC (Digital to Analog Converter)	Converts digital signals to analog output.
I2C (Inter-Integrated Circuit)	Connects with OLED displays, sensors, and EEPROM.
SPI (Serial Peripheral Interface)	Used for fast communication with SD cards and displays.
UART (Universal Asynchronous Receiver-Transmitter)	Communicates with other microcontrollers or serial monitors.
PWM (Pulse Width Modulation)	Controls motor speed, LED brightness, and sound output.

2.1.4 How ESP32 Works in This Project?

1. **Reading Sensor Data:**
    - The ESP32 collects temperature and humidity data from the DHT11 sensor.
2. **Controlling Appliances:**
    - Based on sensor readings, the ESP32 activates/deactivates relays to control home appliances.
3. **Communication with Home Assistant:**
    - The ESP32 sends real-time data to the Home Assistant server over Wi-Fi.
    - The user can view and control appliances via a mobile app or web interface.
4. **Alexa Voice Control:**
    - The ESP32 integrates with Alexa, allowing users to turn ON/OFF appliances via voice commands.
5. **Remote Control via Mobile App:**

- Using the ESPHome firmware, the ESP32 enables remote control of home devices via smartphone.

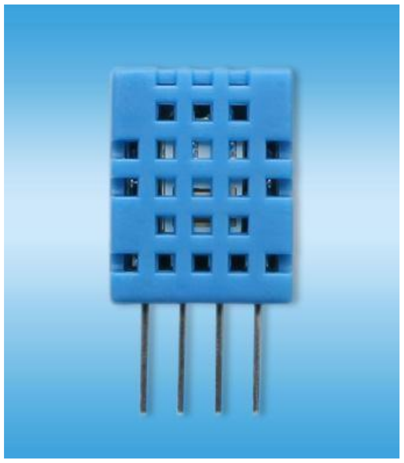
2.1.5 Power Consumption and Sleep Modes

ESP32 is optimized for **low-power operation**, making it ideal for battery-powered smart home devices. It supports the following sleep modes:

1. **Active Mode:** ESP32 is fully operational (~160mA power consumption).
2. **Modem Sleep:** Wi-Fi is disabled, but the CPU remains active (~15mA).
3. **Light Sleep:** CPU and some peripherals are turned off (~2-10mA).
4. **Deep Sleep:** Most functions are disabled except for wake-up triggers (~0.15mA).

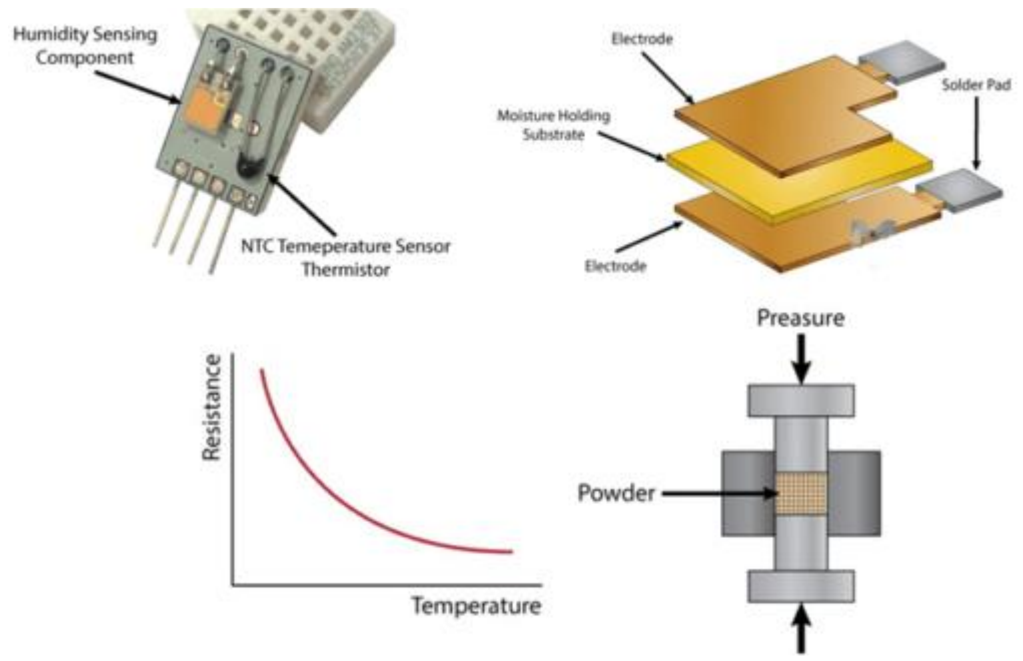
2.2 DHT11 Sensor

DHT11 digital temperature and humidity sensor is a calibrated digital signal output of the temperature and humidity combined sensor. It uses a dedicated digital modules capture technology and the temperature and humidity sensor technology to ensure that products with high reliability and excellent long-term stability. Sensor includes a resistive element and a sense of wet NTC temperature measurement devices, and with a high-performance 8-bit microcontroller connected .



2.2.1 How DHT11 Measures Temperature and Humidity?

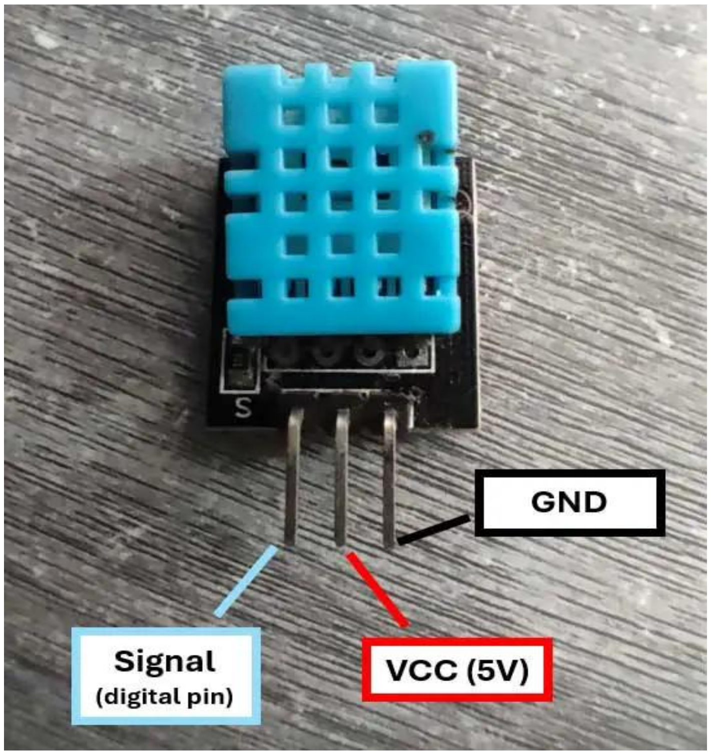
Inside the **DHT11**, there is a thermistor with a humidity sensor component.The humidity sensor component consists of two electrodes containing a dehydrated substrate sandwich. The ions are released into the substrate as water vapor absorbs it, which in turn increases the conductivity between the electrodes.


Humidity Sensing Component
NTC Temeperature Sensor Thermistor
Electrode
Moisture Holding Substrate
Solder Pad
Electrode
Preasure
Resistance
Temperature
Powder

Working of DHT11 Humidity and Temperature Sensor

The resistance change between the two electrodes is proportional to the relative humidity. High relative humidity reduces the resistance between the electrodes, while low relative humidity increases the resistance between the electrodes.DHT11 also includes an NTC / Thermistor to measure temperature. A thermistor is a thermal resistor whose resistance changes rapidly with temperature. The word "NTC" means "**negative temperature coefficient**", which means that the resistance decreases with increasing temperature.

2.2.2 *Features of DHT11:*



1. **Temperature Measurement**
    - Range: **0°C to 50°C**
    - Accuracy: **±2°C**
    - Resolution: **1°C**
2. **Humidity Measurement**
    - Range: **20% to 90% RH (Relative Humidity)**
    - Accuracy: **±5% RH**
    - Resolution: **1% RH**
3. **Operating Voltage**
    - Works on **3.3V or 5V**, making it compatible with most microcontrollers.
4. **Low Power Consumption**
    - Ideal for battery-powered IoT devices.
5. **Digital Output**
    - Uses a **single-wire protocol** for easy communication.

6. Measures **temperature (0–50°C) and humidity (20–90%)** with reasonable accuracy.
7. Provides **digital output** via a single-wire communication protocol.
8. Low-cost and **low power consumption**, making it suitable for continuous monitoring.
9. Compact size for easy integration with microcontrollers like ESP32.

2.2.3 Role in the Project:

- Continuously **monitors room temperature and humidity**.
- Sends temperature data to ESP32 for **automation-based control of appliances**.
- Helps in **energy optimization** by turning devices ON/OFF based on temperature thresholds.
- Displays temperature readings on the **Home Assistant dashboard**.

2.3 4-Channel Relay Module

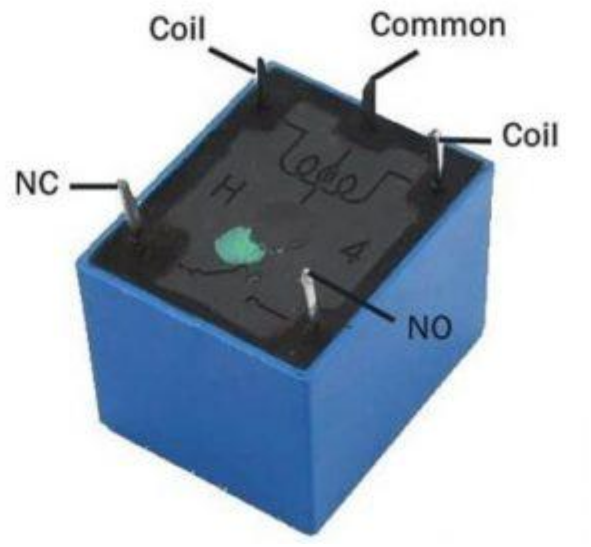
A **relay module** is an electronic switch that allows low-power microcontrollers like ESP32 to control high-power devices such as fans, lights, and ACs. The **4-channel relay module** in this project enables the control of multiple appliances.Relay diagram and explain

2.3.1 What is a 5V Relay?

A 5v relay is an automatic switch that is commonly used in an automatic control circuit and to control a high-current using a low-current signal. The input voltage of the relay signal ranges from 0 to 5V.

2.3.2 5V Relay Pin Configuration

The pin configuration of the 5V relay is shown below. This relay includes 5-pins where each pin and its functionality are shown below.



Relay Pin Diagram

Pin1 (End 1): It is used to activate the relay; usually this pin one end is connected to 5Volts whereas another end is connected to the ground.

Pin2 (End 2): This pin is used to activate the Relay.

Pin3 (Common (COM)): This pin is connected to the main terminal of the Load to make it active.

Pin4 (Normally Closed (NC)): This second terminal of the load is connected to either NC/ NO pins. If this pin is connected to the load then it will be ON before the switch.

Pin5 (Normally Open (NO)): If the second terminal of the load is allied to the NO pin, then the load will be turned off before the switch.

2.3.3 Features

The features of the 5V relay **include the following.**

- Normal Voltage is 5V DC
- Normal Current is 70mA
- AC load current Max is 10A at 250VAC or 125V AC
- DC load current Max is 10A at 30V DC or 28V DC
- It includes 5-pins & designed with plastic material
- Operating time is 10msec
- Release time is 5msec

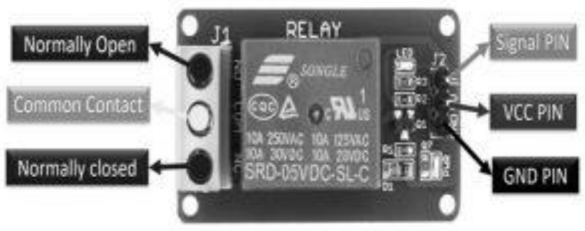
- Maximum switching is 300 operating per minute

2.3.4 5V Relay Module

The relay module with a single channel board is used to manage high voltage, current loads like solenoid valves, motor, AC load & lamps. This module is mainly designed to interface through different microcontrollers like PIC, Arduino, etc.

2.3.5 5V Relay Module Pin Configuration

The pin configuration of the 5V relay module is shown below. This module includes 6-pins where each pin and its functionality are discussed below.



Relay Module Pin Diagram

Normally Open (NO): This pin is normally open unless we provide a signal to the relay modules signal pin. So, the common contact pin smashes its link through the NC pin to make a connection through the NO pin

Common Contact: This pin is used to connect through the load that we desire to switch by using the module.

Normally Closed (NC): This NC pin is connected through the COM pin to form a closed circuit. However, this NC connection will break once the relay is switched through providing an active high/low signal toward the signal pin from a microcontroller.

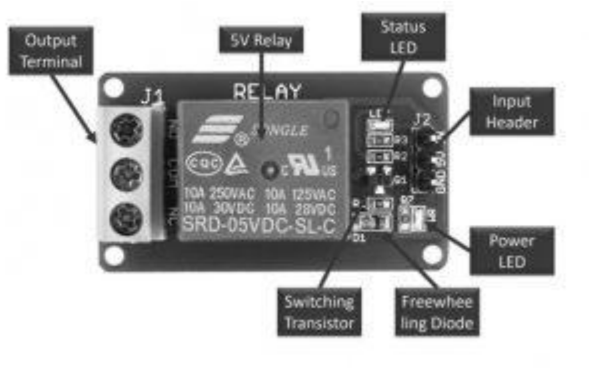
Signal Pin: The signal pin is mainly used for controlling the relay. This pin works in two cases like active low otherwise active high. So, in active low case, the relay activates once we provide an active low signal toward the signal pin, whereas, in an active high case, the relay will trigger once we provide a high signal toward the signal pin.

However, these modules generally work on an active high signal which will strengthen the relay coil to make contact with the common terminal with the normally open terminal.

5V VCC: This pin needs 5V DC to work. So 5V DC power supply is provided to this pin.
Ground: This pin connects the GND terminal of the power supply.

2.3.6 5Volts 1-Channel Relay Module Components

The components in a 5v relay module with a single channel include a relay, output terminal, status LED, power LED, freewheeling diode, input connector & switching transistor.



Relay Module Components

3 Relay

A 5V relay is coated with blue color plastic material. For both AC & DC loads, the utmost operating voltage & current are also displayed on the relay. This relay operates with 5V, so it is called a 5V relay.

4 Output Terminal

The output terminal of the relay module is located at the left-hand side, used to fix an AC/DC load & AC/DC i/p power source. Every o/p connector’s terminal is connected through NC, COM pins & NO of the relay.

The relay module consists of screws that are used to connect wires & cables. The max current supported by this module is 10A & the max contact voltage is 250V AC & 30V DC. Thick main cables are mainly used whenever high voltage & current load is used.

5 Status LED

Status LED is connected by using a current limiting resistor that is located on the top right side of the relay module. So this LED illustrates the relay status by activating the relay & coil through a signal pin. The DC supplies throughout a relay coil.

6 Power LED

Power LED shows the condition of the power source that is connected through the single channel module. If we provide the above 5V source toward both the pins of the module like Vcc & GND, the LED will be damaged due to high voltage.

7 Freewheeling Diode

The connection of this diode can be done across the coil to keep away from the back EMF effect, so-called a flyback diode. The type of coil used in the relay is the inductive type. Once the current supplies throughout an inductive load, then it generates a back EMF voltage, which may harm the circuit. So, this diode is mainly used to keep away from this effect.

8 Input Connector

The input connector is located on the right side of the module. This connector is mainly used to supply a 5V power supply & input signal. In addition, it also supplies power supply toward the power LED, relay coil & status LED.

9 Switching Transistor

Generally, the input signal which is given to a relay is from the I/O pins of microcontrollers like ESP32, TM4C123, Arduino, etc. However, the highest current sourcing capacity of GPIO pins is usually below 20mA**.**

Therefore, a switching transistor is used in this module is to strengthen the current to the requirement of the minimum current level of the relay coil. A switching transistor is used to control the 5V relay from the microcontroller’s GPIO pin.

Some kinds of relay modules are available with an optoisolator like a switching device to give optical isolation among high & low voltage circuits.

However, if you are utilizing a separate relay exclusive of a module & you want to utilize several relays within your projects, then a relay driver IC can be used to drive several arrays from the pins of GPIO in a microcontroller.

2.3.7 Specifications

The specifications of a 1- channel relay module include the following.

- Voltage supply ranges from 3.75V – 6V
- Quiescent current is 2mA
- Once the relay is active then the current is ~70mA
- The highest contact voltage of a relay is 250VAC/30VDC
- The maximum current is 10A

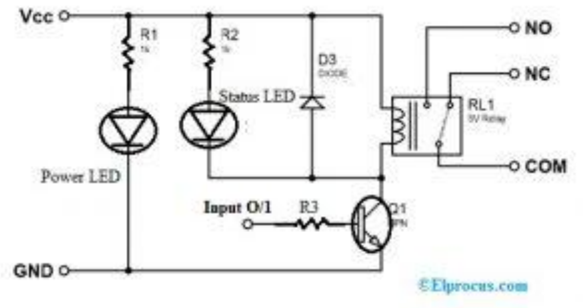
2.3.8 Working

The relay uses the current supply for opening or closing switch contacts. Usually, this can be done through a coil to magnetize the switch contacts & drags them jointly once activated. A spring drives them separately once the coil is not strengthened.

By using this system, there are mainly two benefits, the first one is, the required current for activating the relay is less as compared to the current used by relay contacts for switching. The other benefit is, both the contacts & the coil are isolated galvanically, which means there is no electrical connection among them.

2.3.9 How to Use/Relay Module Circuit Diagram

The circuit diagram of the single-channel relay module circuit is shown below. In this circuit, we can observe that how the relay module is activated and deactivated through a digital signal. This signal is applied to a control pin of the relay module. The following circuit diagram is the internal 5V single channel relay module diagram.



Single Channel Relay Module Circuit

In the above circuit diagram, the single-channel relay module includes resistors-2, transistors, LEDs-2 & a 5V relay. Relay modules are available in two types based on the control signal type used for activation of the relay.

One relay module comes with an NPN transistor whereas another module comes with a PNP transistor. If the relay module uses an NPN Transistor, then it will activate the relay by applying an active high signal to the control pin. Alternatively, if a PNP is used then the relay will be activated through an active low signal on the control pin.
It’s working in proteus simulation software is, when we provide an active high signal toward the control pin in a relay module, then the coil in the relay activates to make the relay active through the connection of the NO pin through the COM pin.

Likewise, once we provide an active low no signal toward the relay’s control pin, then the coil deactivates using a freewheeling diode so that the relay will be deactivated.

In the same way, for PNP based relay module, the relay is activated through an active low signal, whereas an active high signal will deactivate the relay.

The controlling of a 5v single channel relay module can be done by interfacing any kind of microcontroller. For that, we use a GPIO pin like a digital o/p pin which gives an active high & low signal toward the control pin. Once the relay activates, we can listen to an audible sound that comes from the module.

2.3.10 Advantages

The **advantages of the relay module** include the following.

- A remote device can be controlled easily
- It is triggered with less current but it can also trigger high power machines
- Easily contacts can be changed
- At a time, several contacts can be controlled using a single signal
- Activating part can be isolated
- It can switch AC or DC
- At high temperatures, it works very well

2.3.11 Disadvantages

The **disadvantages of the relay module** include the following.

- When contacts of relay modules are used overtime then they may damage
- Noise can be generated through the opening & closing of the contacts.
- Time taken for switching is High

2.3.12 Applications

Relay modules are used in different applications which include the following.

- Used in over voltage/under voltage protection system
- Mains Switching
- Speed control of motors through start-delta converters
- Automatic electrical appliances
- Electrical isolation in between high & low power sources
- Lights
- AC voltage load switching using less voltage DC
- Delivery of Isolated power
- Home automation projects
- Switching with High Current

2.3.13 Role in the Project:

- Controls **home appliances** based on commands from ESP32.
- Enables **automation of lights, fans, and AC** based on temperature, time, or voice commands.
- Provides **manual override** through Home Assistant
- If the **temperature exceeds 35°C**, the **relay will turn ON the AC**
- If the **temperature drops below 30°C**, the **relay will turn OFF the AC**.

2.4 Power Supply and Buck Converter

Since different components in the project require different voltage levels, a **buck converter** is used to step down the voltage from **12V to 5V** for safe operation of the ESP32 and other modules.

2.4.1 Power Requirements of Components:

- **ESP32** – Operates at **3.3V** but can take **5V input** via USB or regulator.

- **Relay Module** – Requires **5V** for activation.
- **Sensors (DHT11, etc.)** – Operate at **3.3V or 5V**.

2.4.2 Role in the Project:

- **Regulates power supply** to prevent damage to ESP32 and other components.
- Ensures stable operation of sensors and actuators.

2.5 Home Assistant OS

Home Assistant is an **open-source home automation platform** that allows users to control and automate smart devices. It provides a centralized interface to monitor and manage IoT-enabled appliances.

2.5.1 Features of Home Assistant:

- **Dashboard for monitoring device status** and controlling appliances.
- **Automation scripts** to trigger actions based on conditions.
- **Integration with Alexa, Google Assistant, and smart home platforms**.
- **Secure and local execution**, ensuring privacy.

2.5.2 Role in the Project:

- Provides an **easy-to-use interface** for controlling home appliances.
- Displays **real-time temperature and humidity** from the DHT11 sensor.
- Enables **scheduling and automation of appliances**.
- Acts as a **bridge between ESP32 and Alexa**.

3.6 Alexa Integration

Amazon Alexa is a **voice assistant** that enables **hands-free control** of smart home devices using voice commands. In this project, Alexa is used to control home appliances via **ESP32 and Home Assistant**.

3.6.1 Features of Alexa Integration:

- **Voice Control** – Turn appliances ON/OFF using simple voice commands.
- **Custom Automation** – Execute complex automation sequences.
- **Remote Access** – Control devices from anywhere using Alexa app.

3.6.2 Role in the Project:

- Enables **voice commands** such as:
  - *"Alexa, turn ON the bedroom fan."*
  - *"Alexa, set AC temperature to 25°C."*
- Provides **seamless integration with Home Assistant**.
- Enhances accessibility and ease of use for users.

3.7 Wi-Fi and Mobile Hotspot Connectivity

A **Wi-Fi connection** is essential for enabling communication between the ESP32, Home Assistant, and Alexa. The system can also be configured to work with a **mobile hotspot** if a home Wi-Fi network is unavailable.

3.7.1 Features of Wi-Fi Connectivity:

- **Allows remote access** to control home appliances from anywhere.
- **Enables real-time data transfer** between ESP32 and Home Assistant.
- **Supports Over-the-Air (OTA) updates**, allowing remote firmware upgrades.

3.7.2 Role in the Project:

- ESP32 connects to **Wi-Fi** to communicate with Home Assistant.
- Home Assistant synchronizes **device states and automation settings**.
- Users can control devices via the **mobile app or Alexa from any location**.

System Design and Implementation

3.1 System Architecture

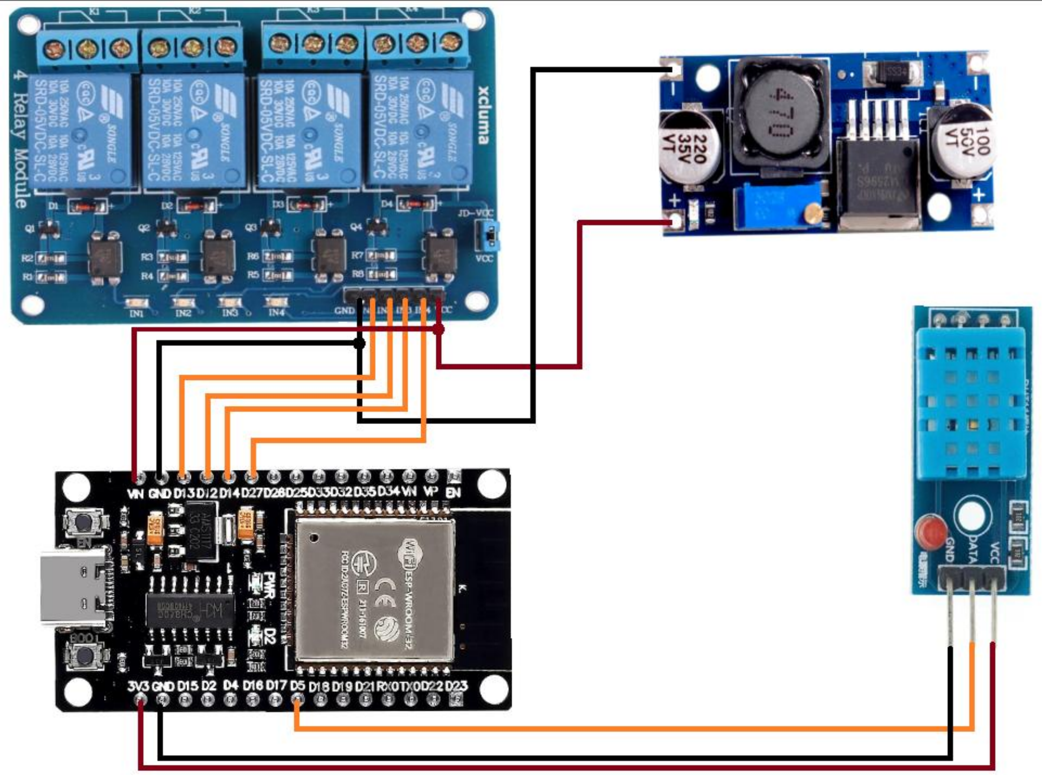
The system architecture outlines how different components communicate and function together. The ESP32 microcontroller serves as the core processing unit, interfacing with the **DHT11 sensor** (for temperature monitoring) and a **4-channel relay module** (for controlling appliances). Data from the ESP32 is sent to **Home Assistant**, where it can be monitored and controlled remotely via a web-based dashboard.

- **ESP32** collects temperature data from the **DHT11 sensor** and determines whether to turn the AC on/off based on a predefined threshold.
- The **4-channel relay module** is used to switch electrical appliances (e.g., AC, lights, fans) ON/OFF based on commands from **ESP32**.
- **ESPHome firmware** runs on ESP32, allowing seamless integration with **Home Assistant**.
- Users can **remotely control appliances** through the Home Assistant dashboard or use **Alexa voice commands** to trigger the relays.
- The ESP32 is connected to **Wi-Fi**, enabling communication with Home Assistant and Alexa.

3.2 Circuit Diagram and Explanation

A **circuit diagram** is essential for illustrating how components are physically connected. The diagram should include:

- **ESP32 connections**
  - GPIO pins connected to **DHT11 sensor** (for temperature input).
  - GPIO pins controlling the **relay module** for AC switching.
- **Power supply connections**
  - ESP32 powered through a **5V power adapter** or **buck converter**.
  - Relays connected to an external **12V power supply** for handling high-power appliances.



- **Explanation:**

- **DHT11 Sensor**: Sends temperature data to ESP32 via a digital pin.
- **Relay Module**: Receives control signals from ESP32 to turn appliances ON/OFF.
- **Power Supply**: Uses a **buck converter** to step down voltage if needed.

3.3 Power Supply Design

- **ESP32 operates at 3.3V**, but can take **5V input via USB or VIN pin**.
- **Relay module requires 5V or 12V**, depending on the model.
- If using a **12V power adapter**, a **buck converter** (step-down module) is needed to convert **12V to 5V** for powering ESP32.
- The **relay module** has an **opto-isolated circuit** to prevent high-voltage spikes from affecting ESP32.

3.4 ESPHome Firmware and Configuration

ESPHome is an open-source firmware that simplifies ESP32 configuration for Home Assistant integration.

- **Why use ESPHome?**
    - It eliminates complex coding, as ESP32 can be programmed using **YAML configuration**.
    - It enables **real-time monitoring** and **automation rules** for relays and sensors.

3.4.1 Steps to Configure ESPHome:

1. Install ESPHome on your **Home Assistant** setup.
2. Connect ESP32 to your PC and flash ESPHome firmware.
3. Define sensor and relay configurations in a **YAML file**.
4. Upload the configuration wirelessly to ESP32.

3.5 Temperature-Based AC Control

The **ESP32 reads temperature data** from the **DHT11 sensor** and makes decisions based on predefined thresholds.

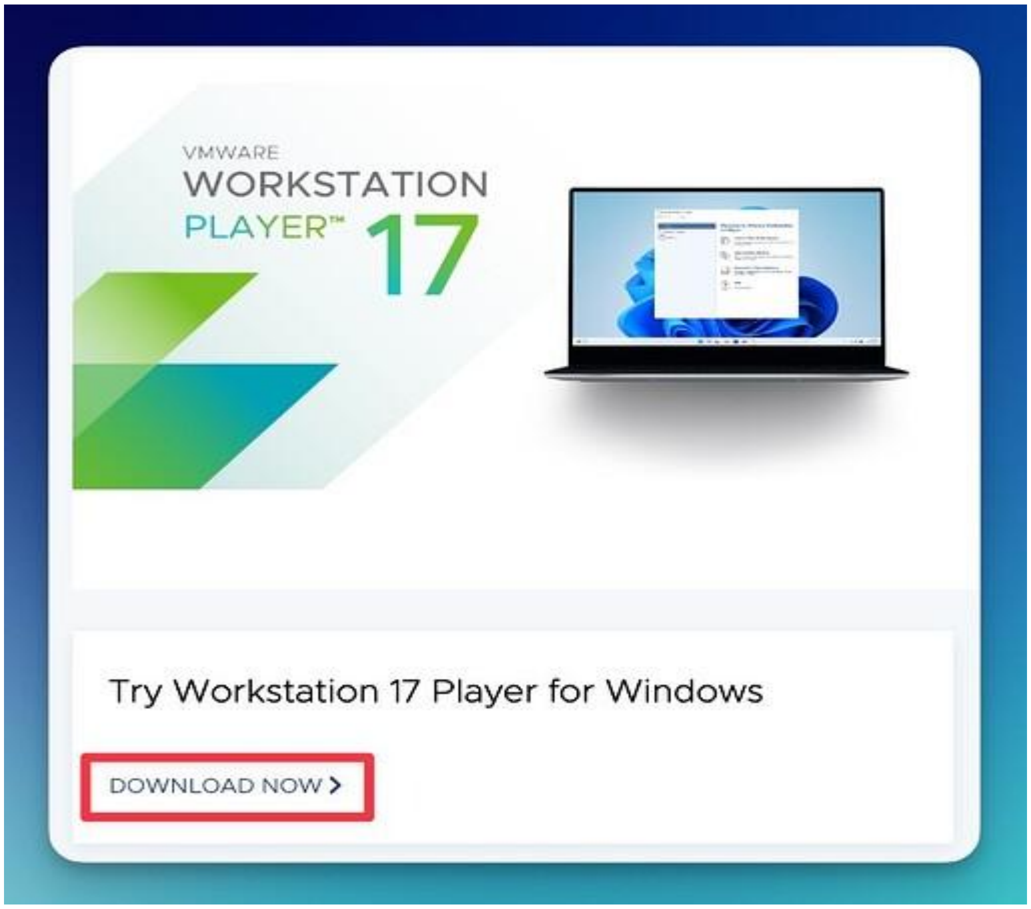
- **Example Automation:**
    - If **temperature > 28°C**, ESP32 **turns AC ON** by activating the relay.
    - If **temperature < 25°C**, ESP32 **turns AC OFF**.
- The temperature-based control logic is written in **ESPHome YAML configuration** or programmed in **Arduino/C++**.
- The **Home Assistant dashboard** can be used to manually override automation.

3.6 Home Assistant Integration

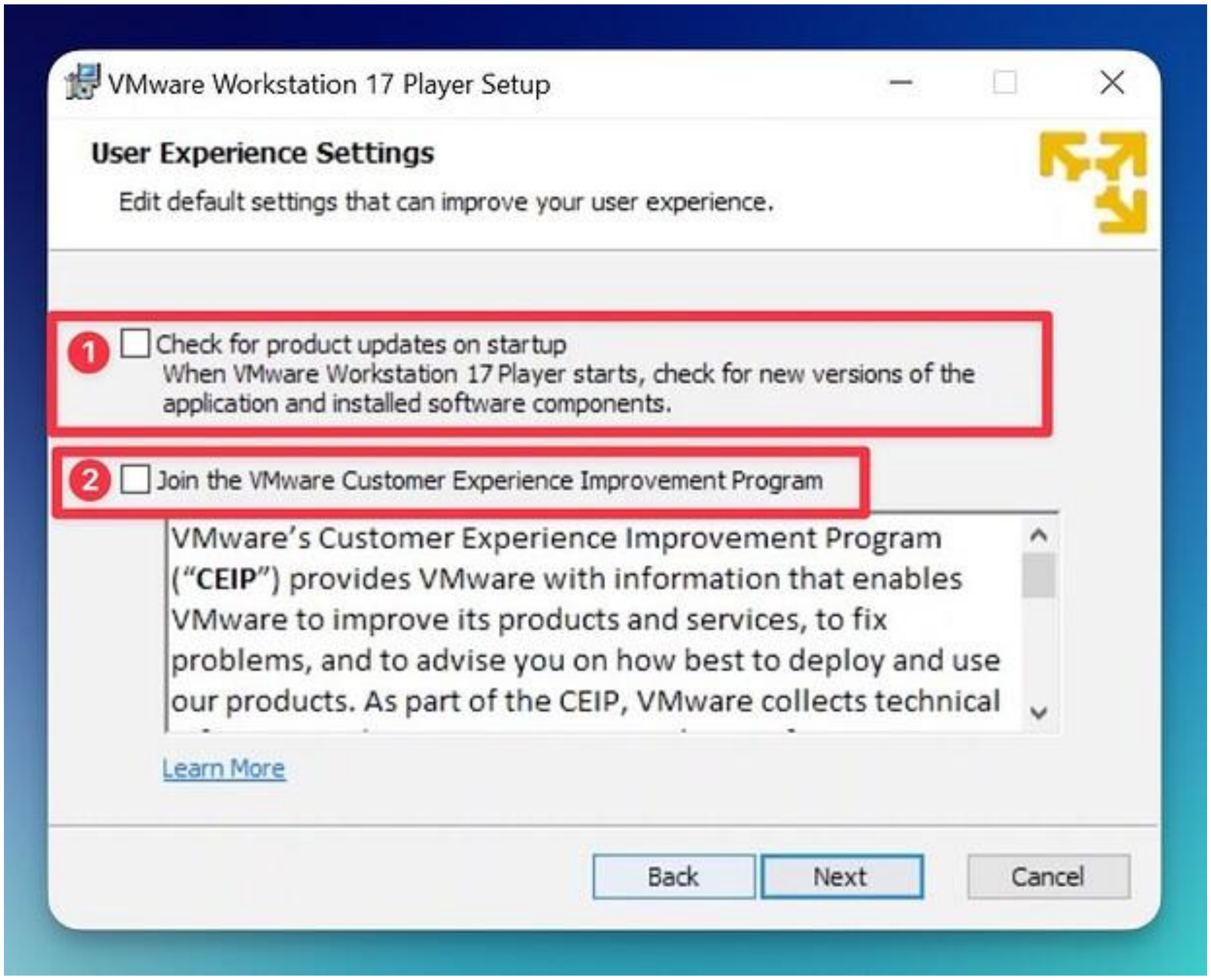
Home Assistant is a powerful home automation platform used for **monitoring and controlling devices remotely**.

Steps for ESP32 Integration with Home Assistant:

1. **Install** Home Assistant OS **on a** VM Player**.**



Then, install VMware 17 Player with all defaults.



VMware Workstation 17 Player Setup

User Experience Settings

Edit default settings that can improve your user experience.

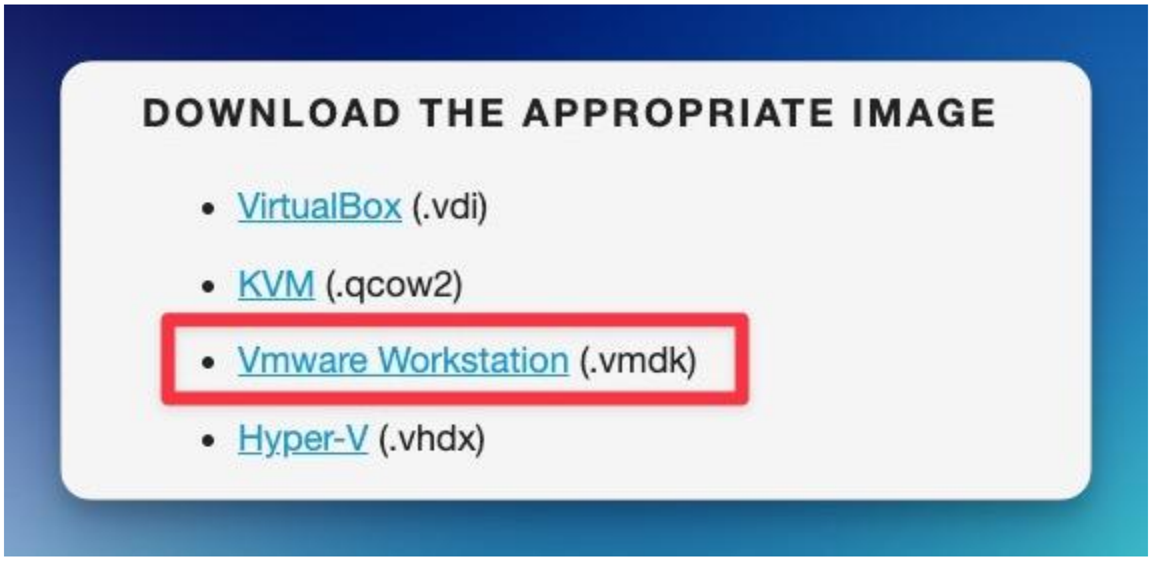
1 ☐ Check for product updates on startup
When VMware Workstation 17 Player starts, check for new versions of the application and installed software components.

2 ☐ Join the VMware Customer Experience Improvement Program

VMware's Customer Experience Improvement Program ("**CEIP**") provides VMware with information that enables VMware to improve its products and services, to fix problems, and to advise you on how best to deploy and use our products. As part of the CEIP, VMware collects technical

Continue with the install until complete.

Download Home Assistant Image
Download the Vmware (.vmdk) image from

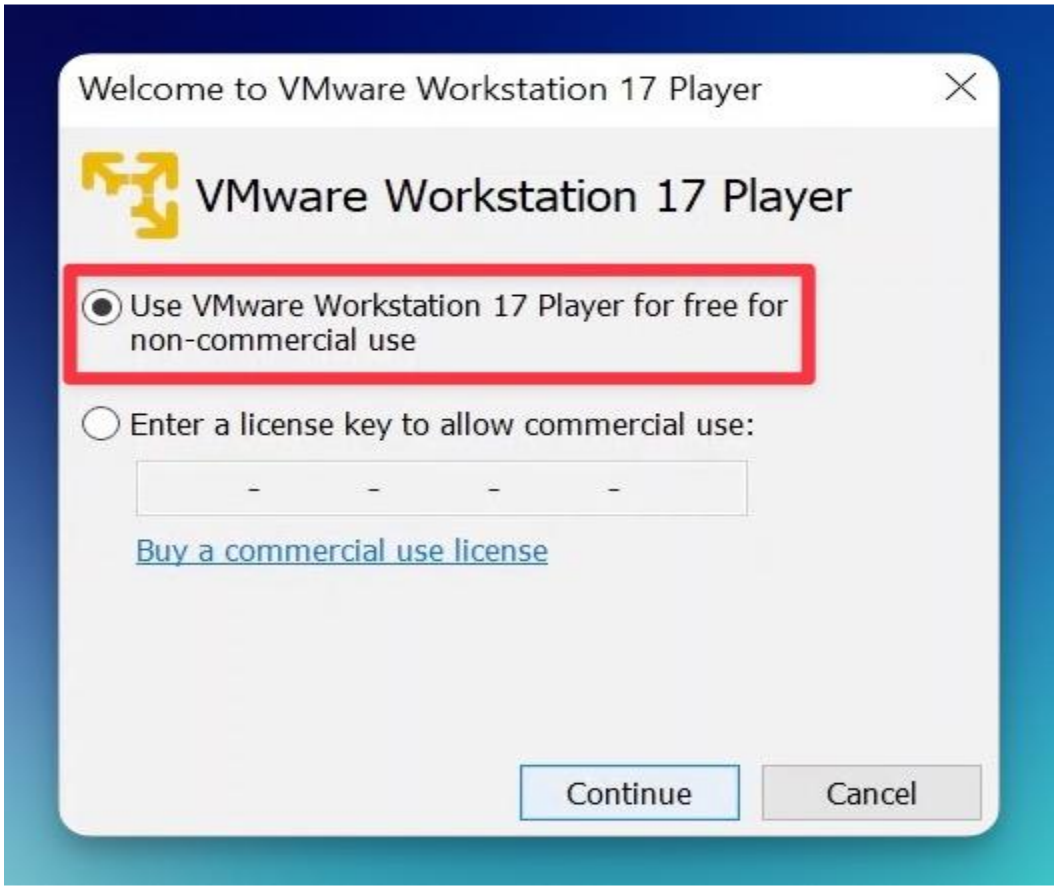

DOWNLOAD THE APPROPRIATE IMAGE

- VirtualBox (.vdi)
- KVM (.qcow2)
- Vmware Workstation (.vmdk)
- Hyper-V (.vhdx)

Extract the file, and place it into a directory where your VM will live (C:\Home Assistant)

Launch VMware 17 Player

Launch **VMware 17 Player**. Select the free for non-commercial use option.


Welcome to VMware Workstation 17 Player

VMware Workstation 17 Player

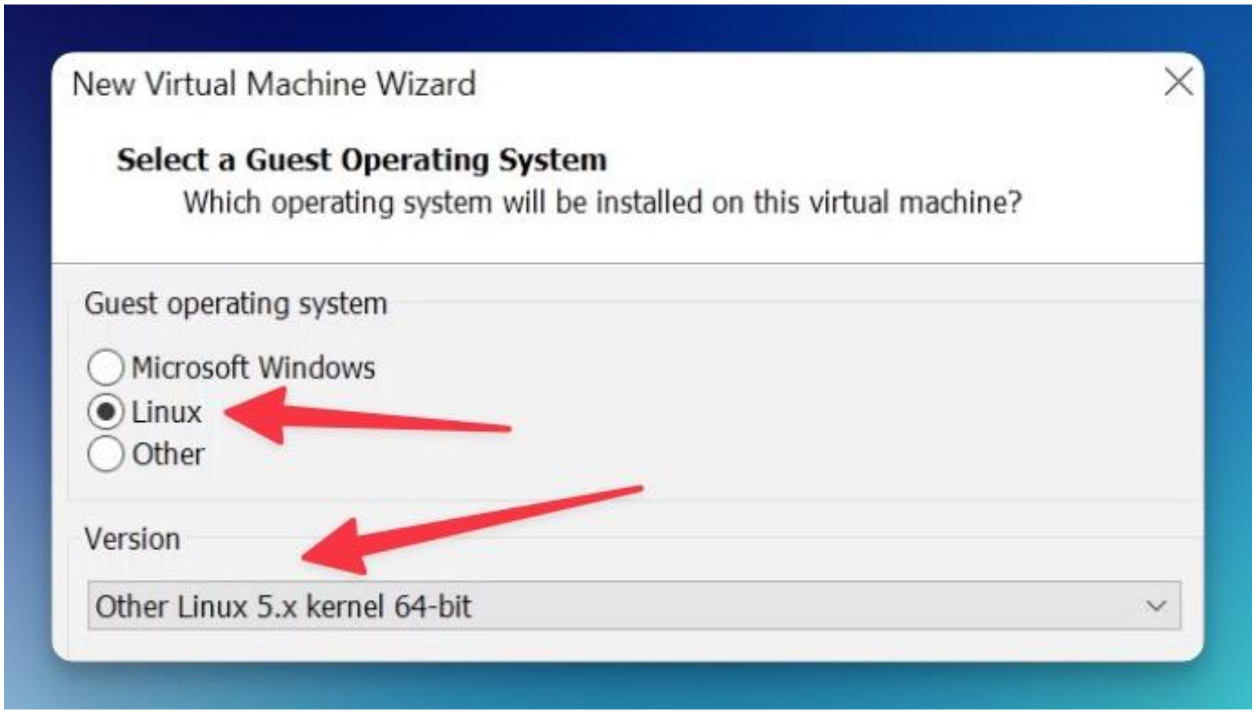
[✓ Use VMware Workstation 17 Player for free for non-commercial use]

Enter a license key to allow commercial use:

Click **Create a New Virtual Machine**

Click **I will install the operating system later**.

Select **Linux**. For the version, you would use **Other Linux 5.x and later kernel 64-bit**.



New Virtual Machine Wizard

Select a Guest Operating System

Which operating system will be installed on this virtual machine?

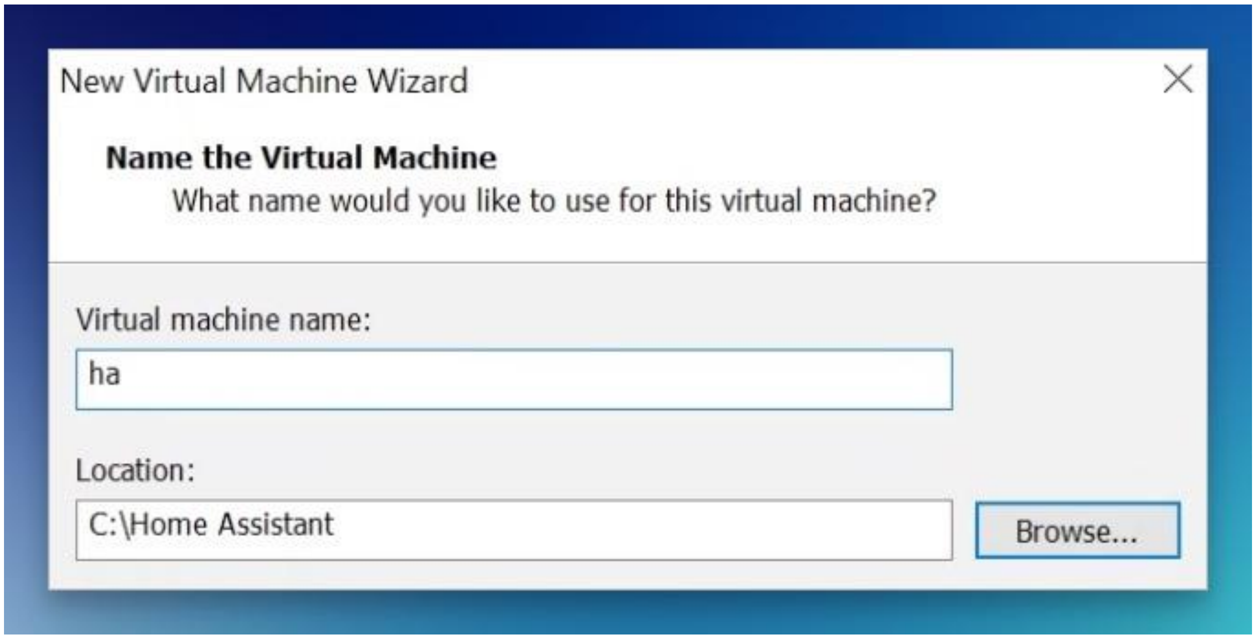
Guest operating system

○ Microsoft Windows
● Linux
○ Other

Version

Other Linux 5.x kernel 64-bit

Give your VM a name (for example: **ha**) and select the folder you created in the previous step.



New Virtual Machine Wizard

Name the Virtual Machine

What name would you like to use for this virtual machine?

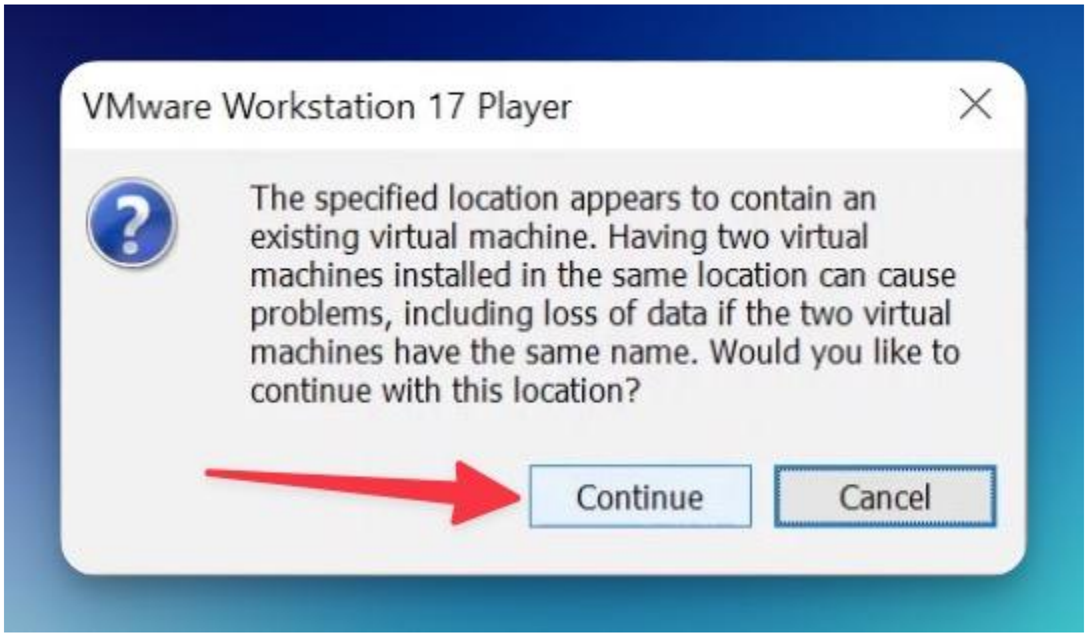
Virtual machine name:

ha

Location:

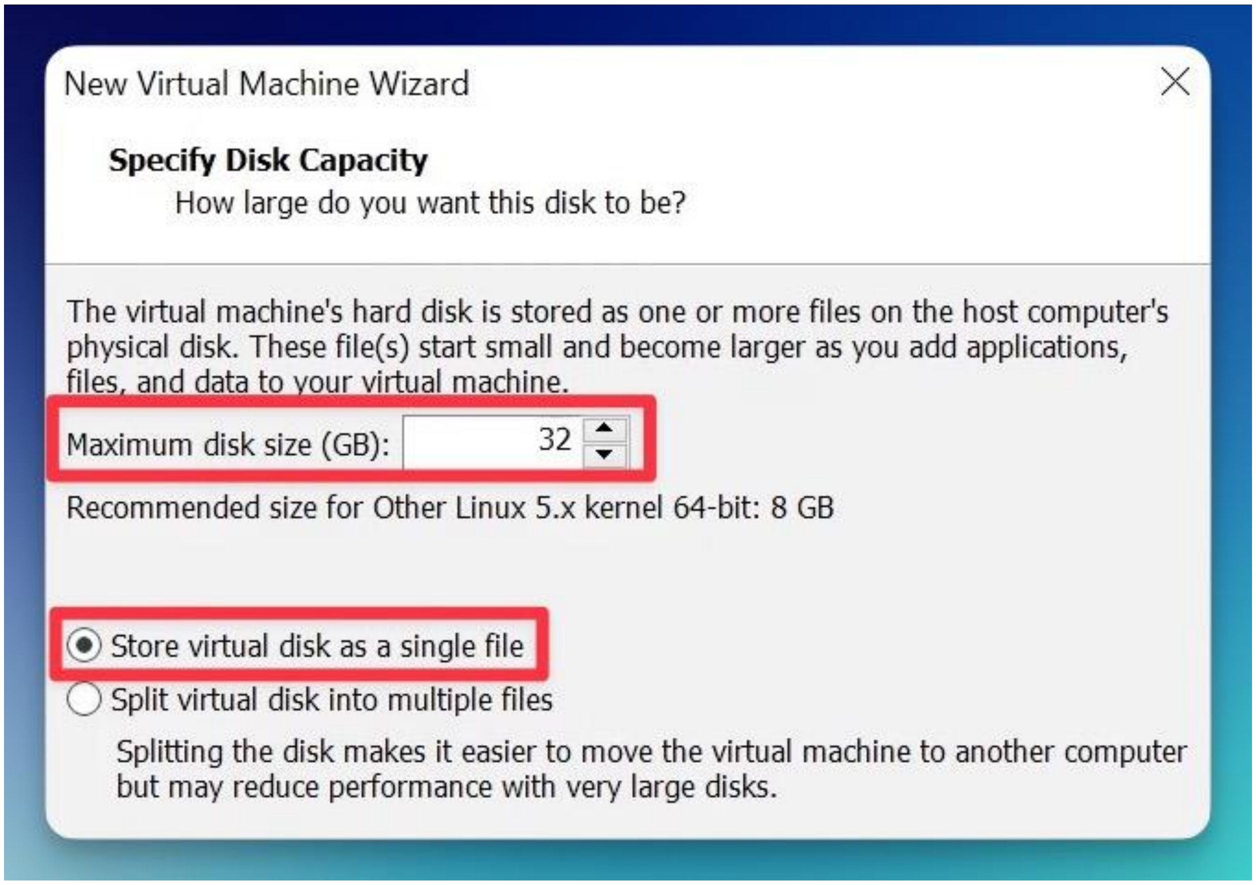
C:\Home Assistant

Confirm the popup. Select **Continue**.


VMware Workstation 17 Player

The specified location appears to contain an existing virtual machine. Having two virtual machines installed in the same location can cause problems, including loss of data if the two virtual machines have the same name. Would you like to continue with this location?

Store it as a single file, and set your storage size to 32GB as recommended by Home Assistant documentation.


New Virtual Machine Wizard

Specify Disk Capacity
How large do you want this disk to be?

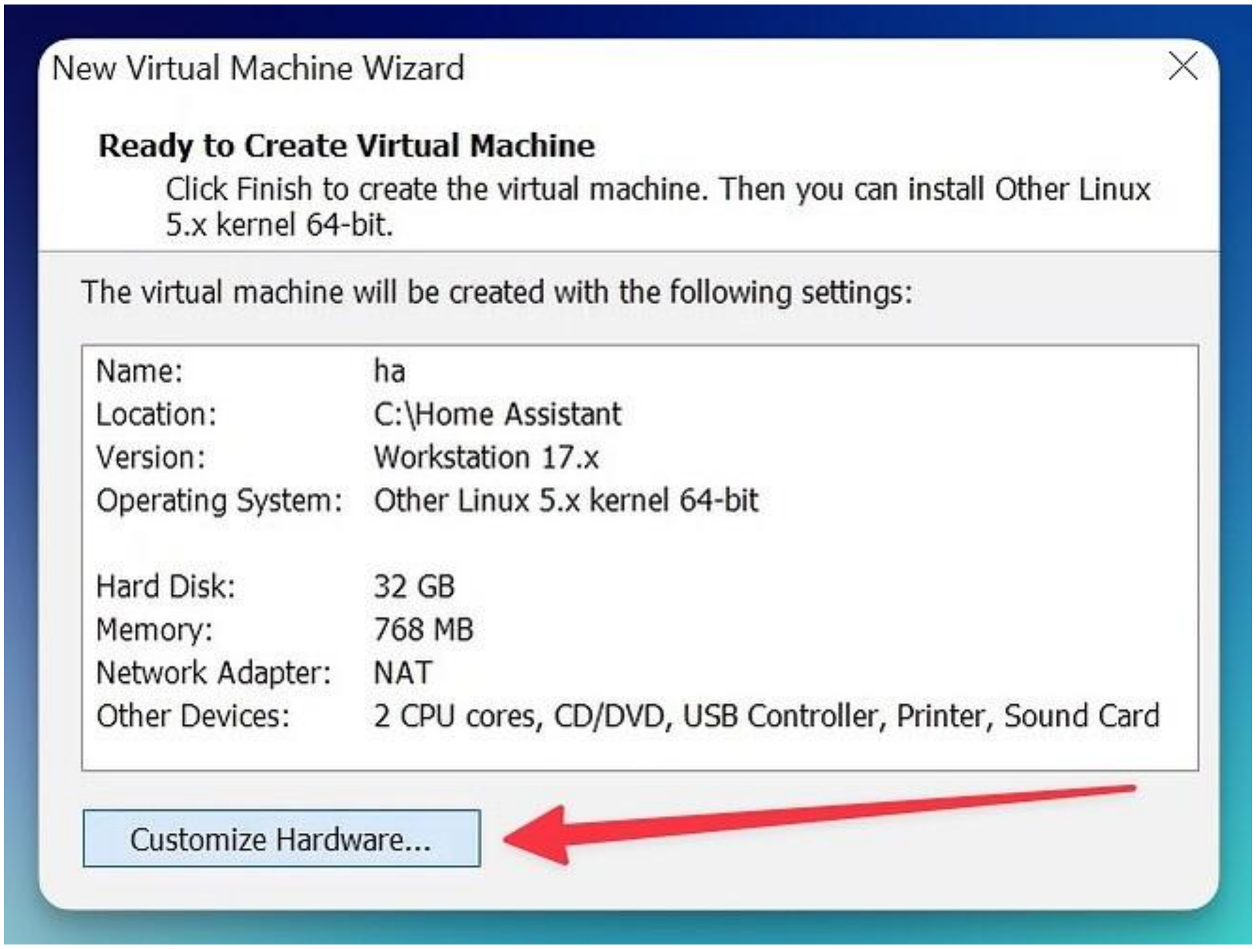
The virtual machine's hard disk is stored as one or more files on the host computer's physical disk. These file(s) start small and become larger as you add applications, files, and data to your virtual machine.

Maximum disk size (GB): 32

Recommended size for Other Linux 5.x kernel 64-bit: 8 GB

[✓ Store virtual disk as a single file]
Split virtual disk into multiple files
Splitting the disk makes it easier to move the virtual machine to another computer but may reduce performance with very large disks.

On the next screen, click **Customize Hardware**.



New Virtual Machine Wizard

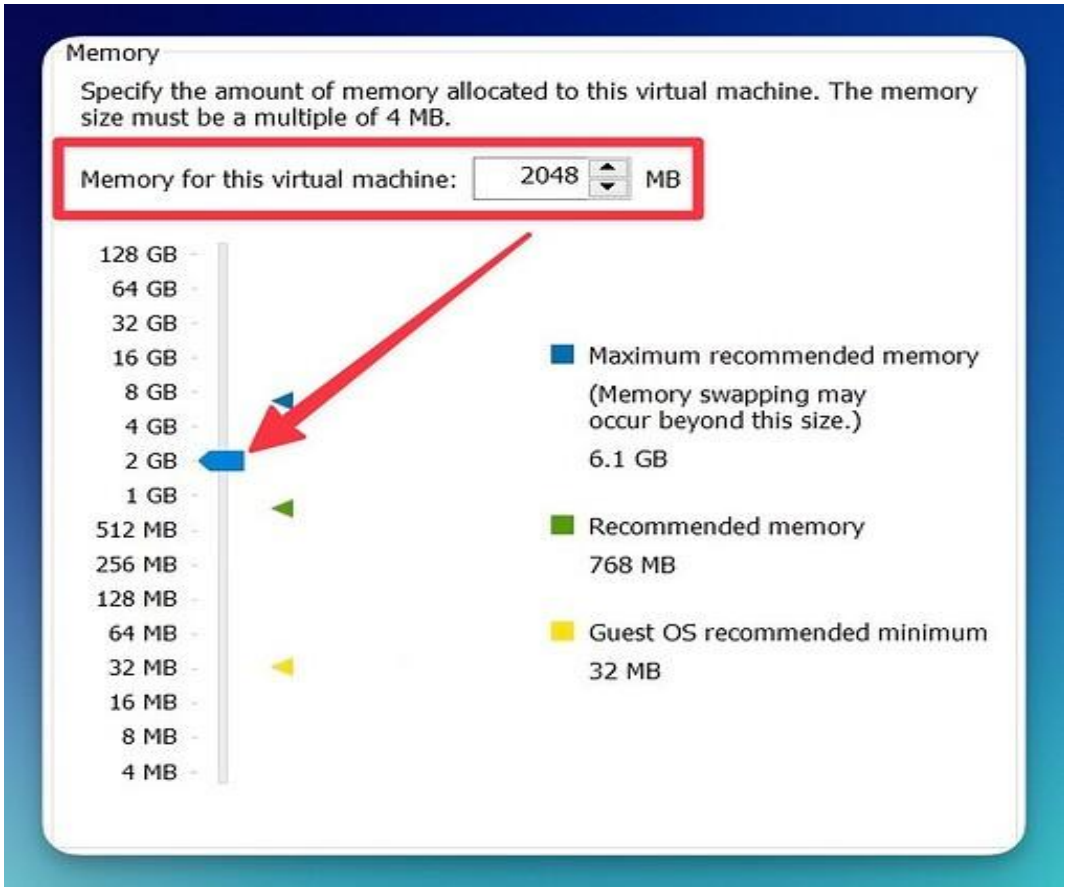
Ready to Create Virtual Machine

Click Finish to create the virtual machine. Then you can install Other Linux 5.x kernel 64-bit.

The virtual machine will be created with the following settings:

Name:	ha
Location:	C:\Home Assistant
Version:	Workstation 17.x
Operating System:	Other Linux 5.x kernel 64-bit
Hard Disk:	32 GB
Memory:	768 MB
Network Adapter:	NAT
Other Devices:	2 CPU cores, CD/DVD, USB Controller, Printer, Sound Card

Set memory to at least 2GB


Memory

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

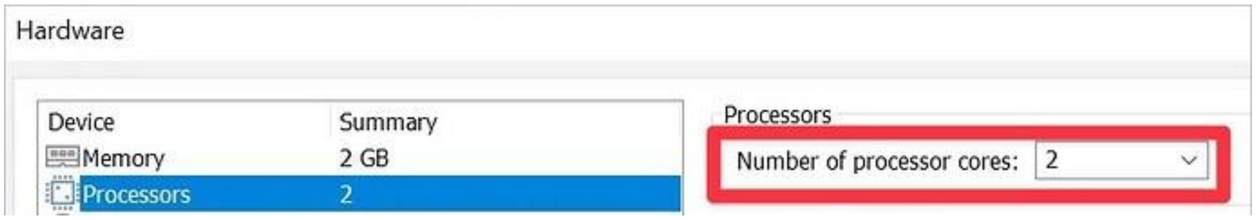
Memory for this virtual machine: 2048 MB

🟦 Maximum recommended memory
(Memory swapping may occur beyond this size.)
6.1 GB

🟩 Recommended memory
768 MB

🟨 Guest OS recommended minimum
32 MB

Make sure that **Processors** are set to at least 2

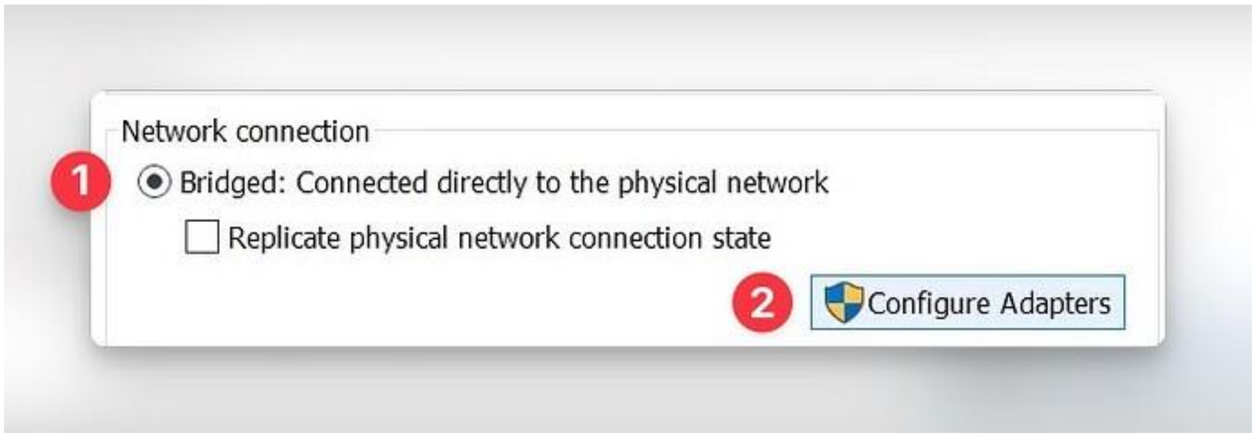

Hardware

Device	Summary
Memory	2 GB
Processors	2

Processors

Number of processor cores: 2

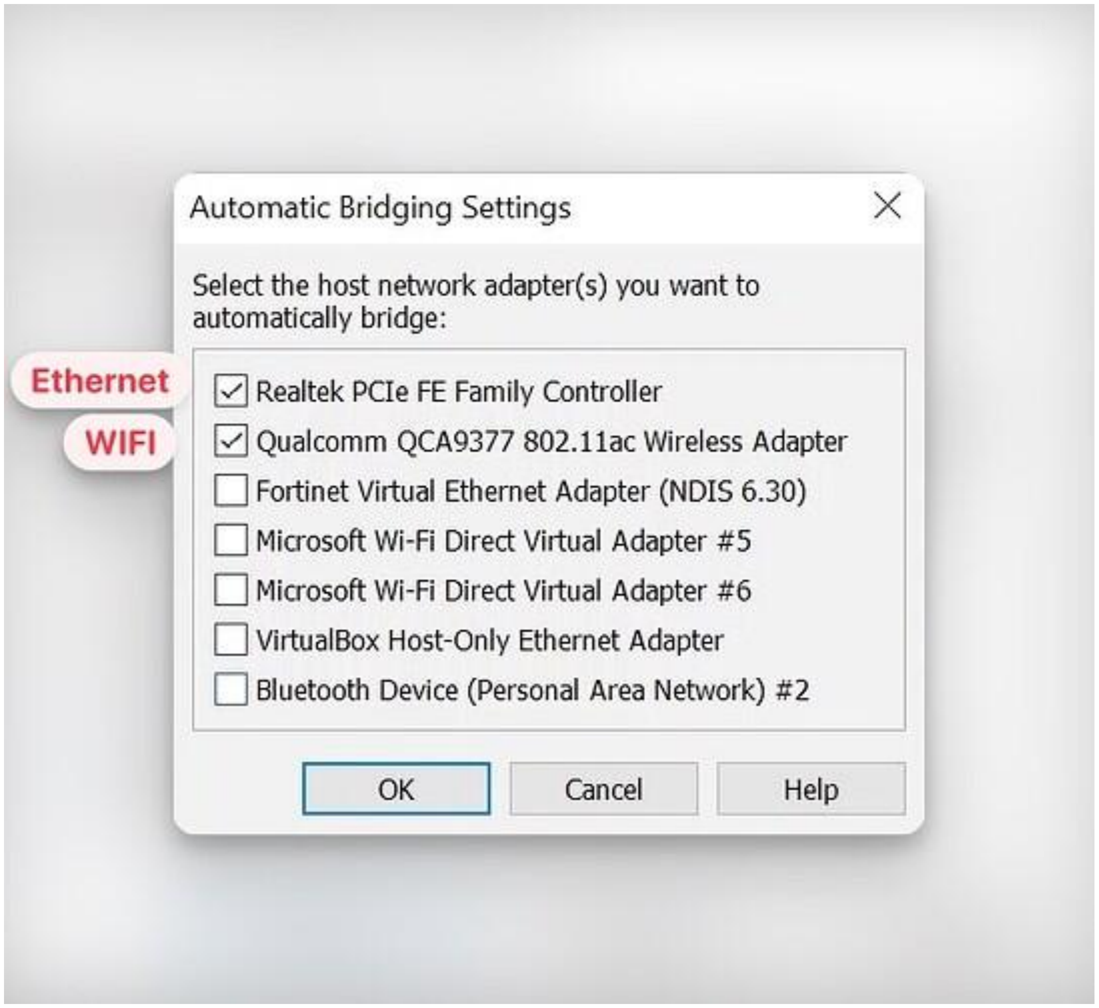
Set the network adapter needs to “bridged” mode.
Then go to **Configure Adapters**


Network connection

1 [● Bridged: Connected directly to the physical network]

☐ Replicate physical network connection state

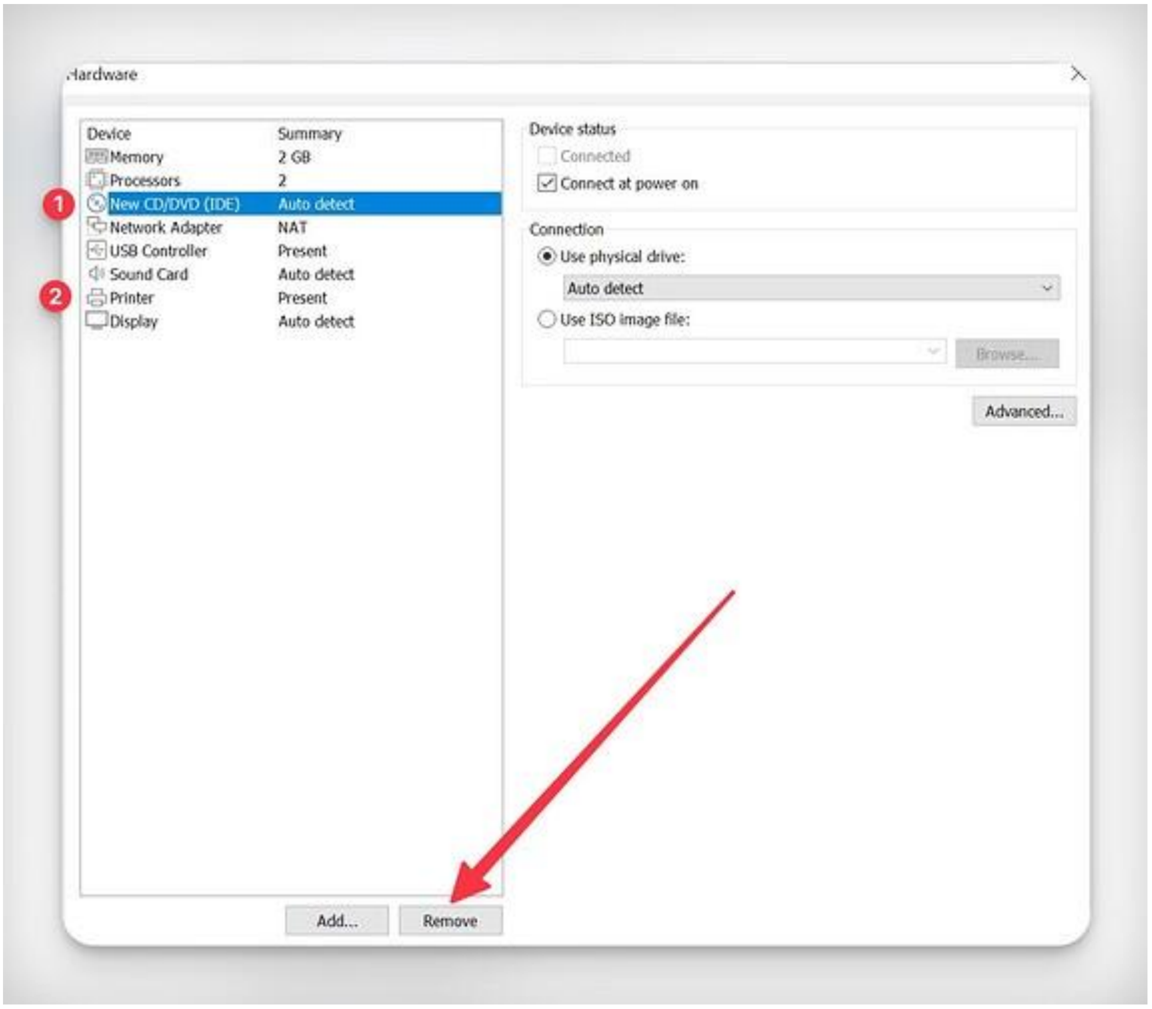
Make sure to select only your Ethernet and WIFI adapters (if you're going to use WIFI)


Automatic Bridging Settings

Select the host network adapter(s) you want to automatically bridge:

- [✓] Realtek PCIe FE Family Controller
- [✓] Qualcomm QCA9377 802.11ac Wireless Adapter
- [ ] Fortinet Virtual Ethernet Adapter (NDIS 6.30)
- [ ] Microsoft Wi-Fi Direct Virtual Adapter #5
- [ ] Microsoft Wi-Fi Direct Virtual Adapter #6
- [ ] VirtualBox Host-Only Ethernet Adapter
- [ ] Bluetooth Device (Personal Area Network) #2

Remove both New CD/DVD (IDE) and Printer as they will not be used.



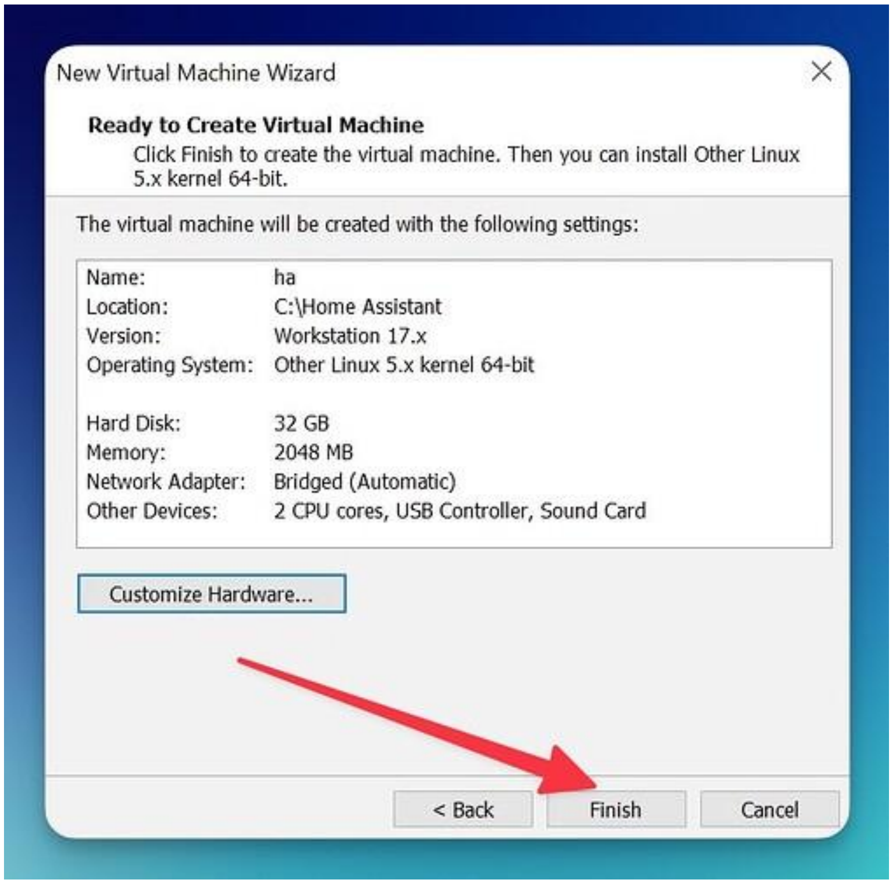
Hardware

Device	Summary
Memory	2 GB
Processors	2
New CD/DVD (IDE)	Auto detect
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	Auto detect

Device status
- [ ] Connected
- [✓] Connect at power on

Connection
- (●) Use physical drive: Auto detect
- ( ) Use ISO image file:

This is how your hardware should look go to Finish to create your Virtual Machine.


New Virtual Machine Wizard

Ready to Create Virtual Machine

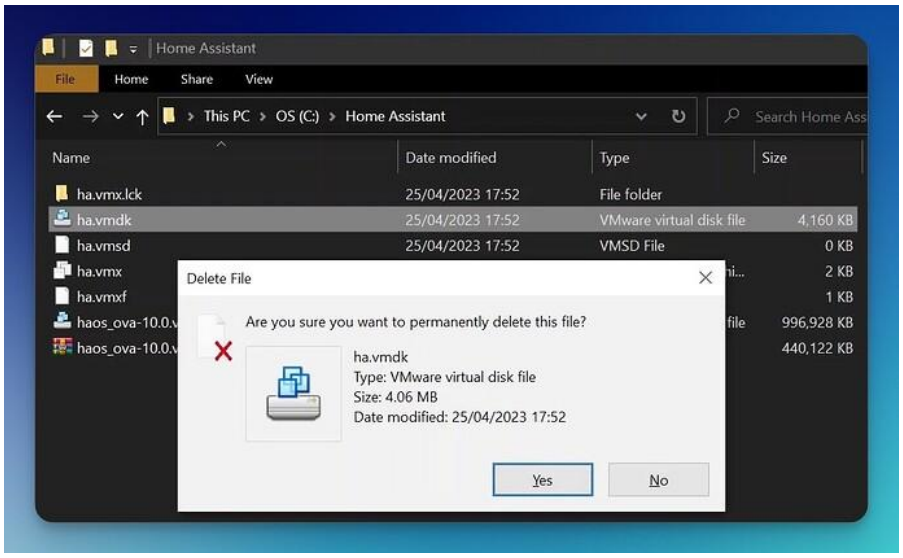
Click Finish to create the virtual machine. Then you can install Other Linux 5.x kernel 64-bit.

The virtual machine will be created with the following settings:

Name:	ha
Location:	C:\Home Assistant
Version:	Workstation 17.x
Operating System:	Other Linux 5.x kernel 64-bit
Hard Disk:	32 GB
Memory:	2048 MB
Network Adapter:	Bridged (Automatic)
Other Devices:	2 CPU cores, USB Controller, Sound Card

Edit VM Settings

Navigate back to the C:\Home Assistant folder. Delete the ha.vmdk file.



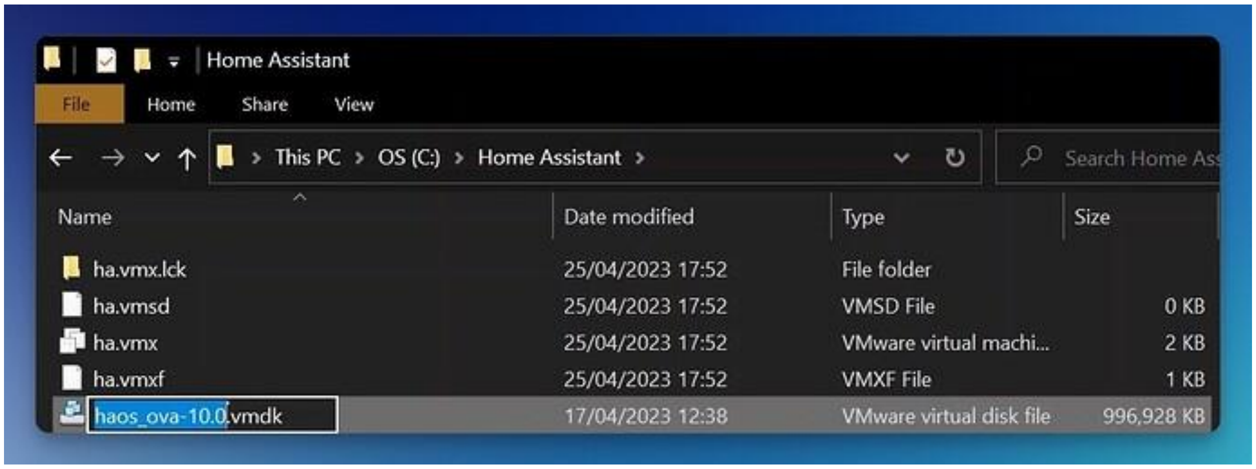
Name	Date modified	Type	Size
ha.vmx.lck	25/04/2023 17:52	File folder	
ha.vmdk	25/04/2023 17:52	VMware virtual disk file	4,160 KB
ha.vmsd	25/04/2023 17:52	VMSD File	0 KB
ha.vmx			2 KB
ha.vmxf			1 KB
haos_ova-10.0.		file	996,928 KB
haos_ova-10.0.			440,122 KB

Delete File

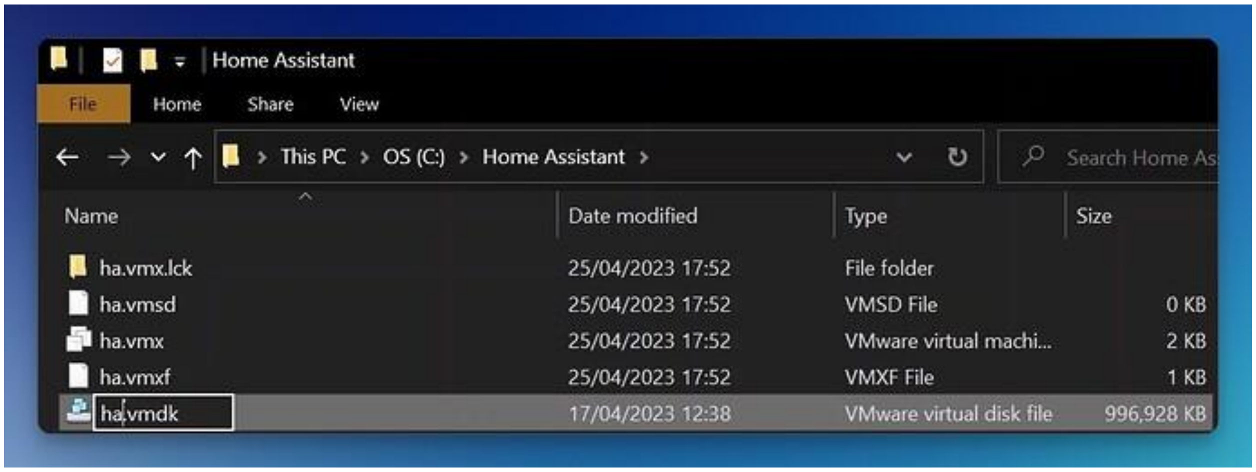
Are you sure you want to permanently delete this file?

ha.vmdk
Type: VMware virtual disk file
Size: 4.06 MB
Date modified: 25/04/2023 17:52

Then **rename** the haos _ova- 10.0.vmdk to ha.vmdk (or the Virtual Machine name you created in the previous step).

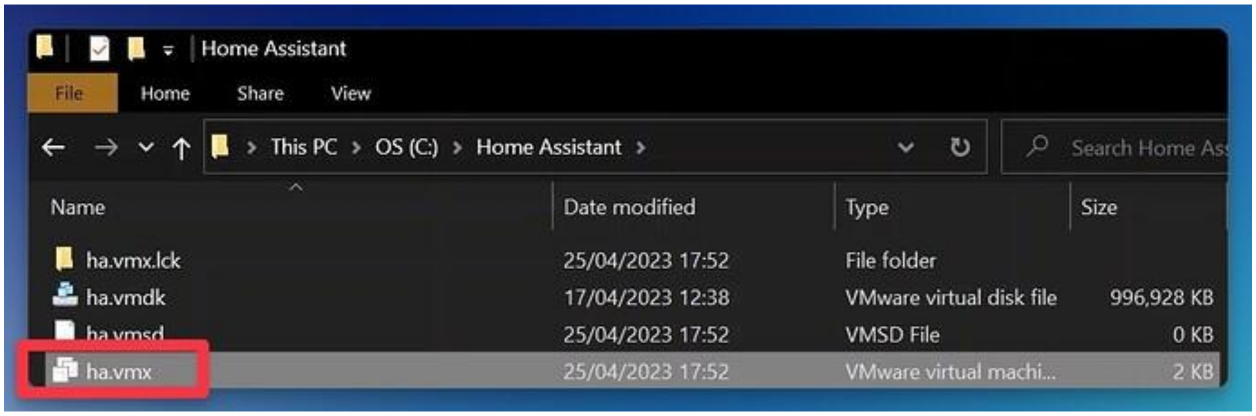


Name	Date modified	Type	Size
ha.vmx.lck	25/04/2023 17:52	File folder	
ha.vmsd	25/04/2023 17:52	VMSD File	0 KB
ha.vmx	25/04/2023 17:52	VMware virtual machi...	2 KB
ha.vmxf	25/04/2023 17:52	VMXF File	1 KB
haos_ova-10.0.vmdk	17/04/2023 12:38	VMware virtual disk file	996,928 KB

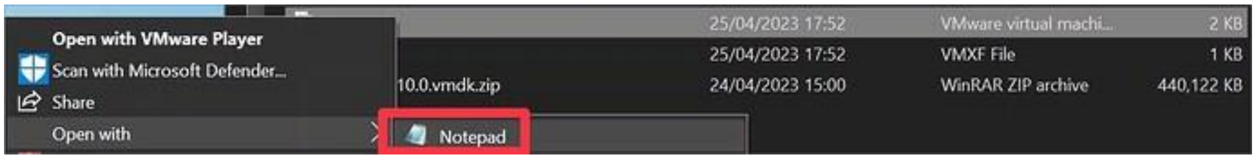


Name	Date modified	Type	Size
ha.vmx.lck	25/04/2023 17:52	File folder	
ha.vmsd	25/04/2023 17:52	VMSD File	0 KB
ha.vmx	25/04/2023 17:52	VMware virtual machi...	2 KB
ha.vmxf	25/04/2023 17:52	VMXF File	1 KB
ha.vmdk	17/04/2023 12:38	VMware virtual disk file	996,928 KB

Then, locate the homeassistant VMware virtual machine configuration file (.vmx). Right-click > Open With > notepad.



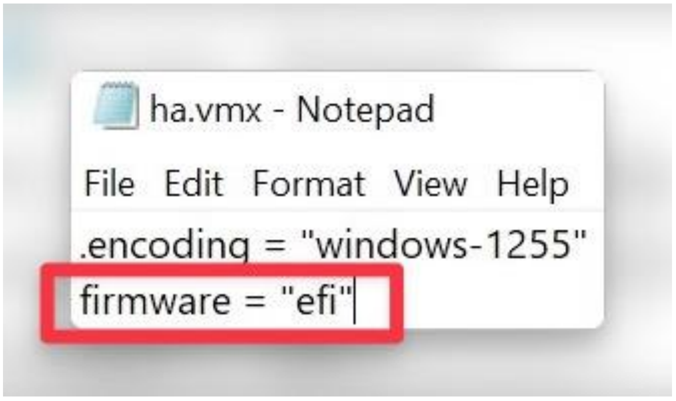
Name	Date modified	Type	Size
ha.vmx.lck	25/04/2023 17:52	File folder	
ha.vmdk	17/04/2023 12:38	VMware virtual disk file	996,928 KB
ha.vmsd	25/04/2023 17:52	VMSD File	0 KB
ha.vmx	25/04/2023 17:52	VMware virtual machi...	2 KB



	25/04/2023 17:52	VMware virtual machi...	2 KB
	25/04/2023 17:52	VMXF File	1 KB
10.0.vmdk.zip	24/04/2023 15:00	WinRAR ZIP archive	440,122 KB

Under the .encoding line, add the following code and save the file.

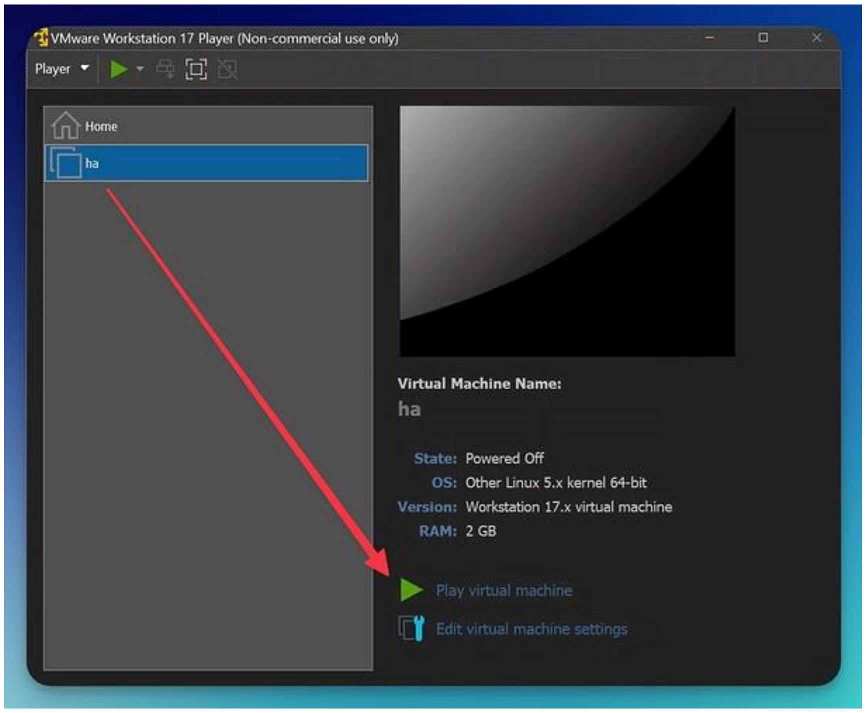
```
firmware="efi"
```



```
.encoding = "windows-1255"
firmware = "efi"
```

Start the Virtual Machine

Select your VMThen, click **Play Virtual machine** or click the green triangle at the top of the window.

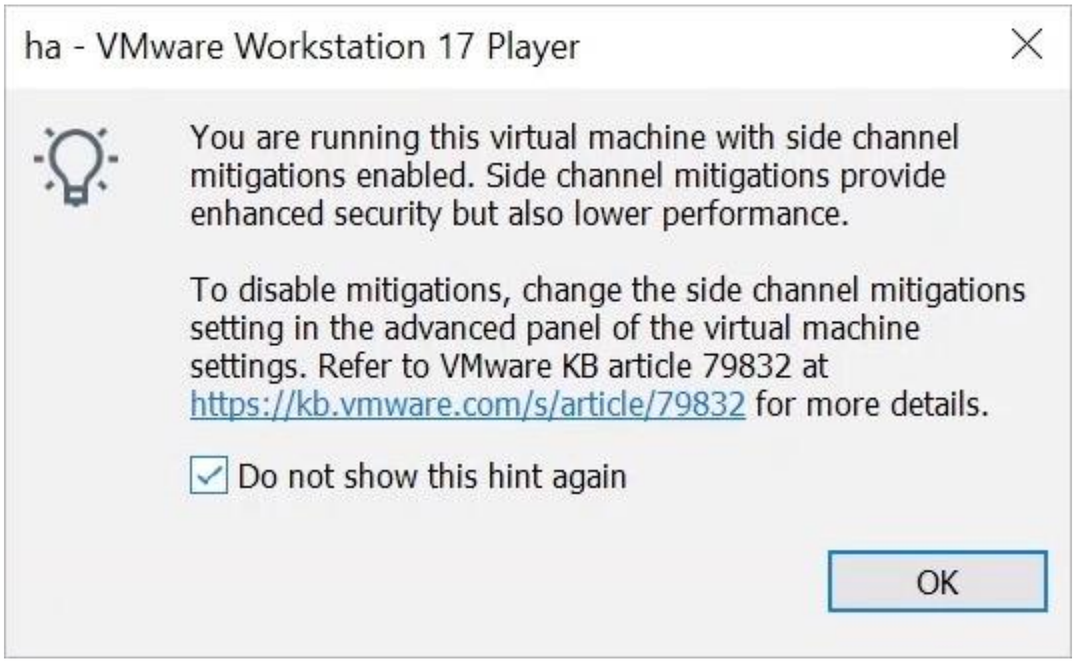

VMware Workstation 17 Player (Non-commercial use only)
Player
Home
ha

Virtual Machine Name:
ha

State: Powered Off
OS: Other Linux 5.x kernel 64-bit
Version: Workstation 17.x virtual machine
RAM: 2 GB

Play virtual machine
Edit virtual machine settings

If you see this message, click OK.

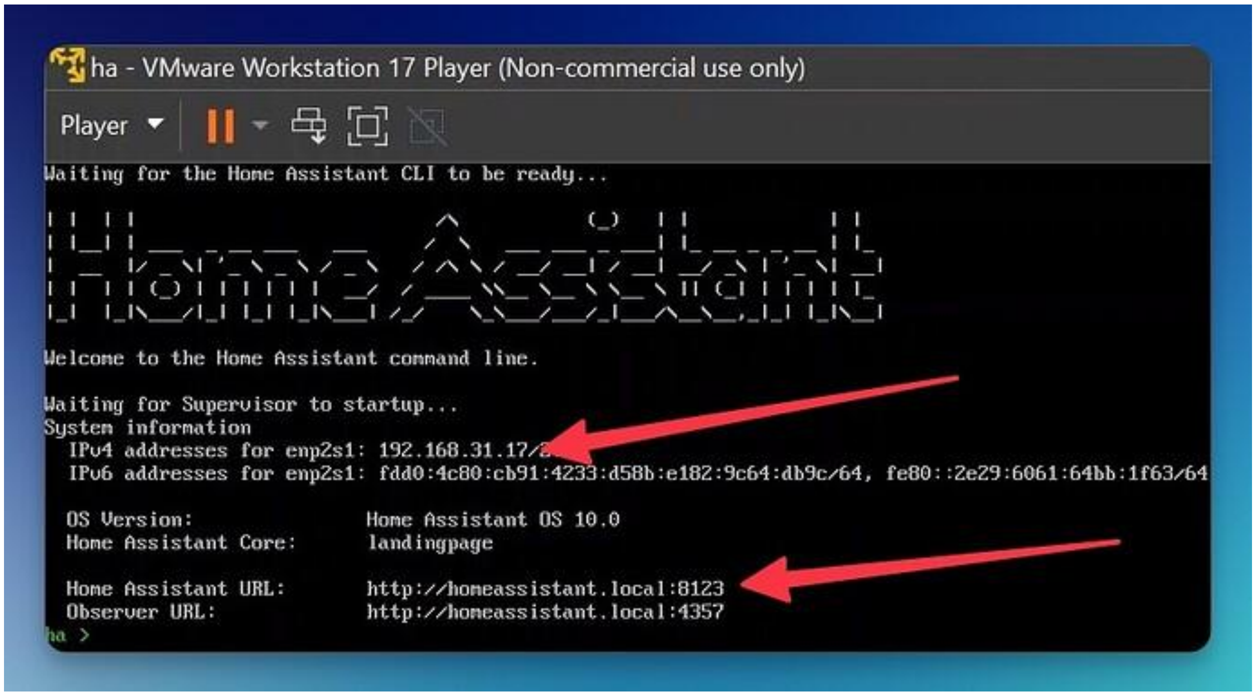

ha - VMware Workstation 17 Player

You are running this virtual machine with side channel mitigations enabled. Side channel mitigations provide enhanced security but also lower performance.

To disable mitigations, change the side channel mitigations setting in the advanced panel of the virtual machine settings. Refer to VMware KB article 79832 at https://kb.vmware.com/s/article/79832 for more details.

[✓] Do not show this hint again

If everything is successful up until this point, you should see the Home Assistant login screen with some IP addresses.



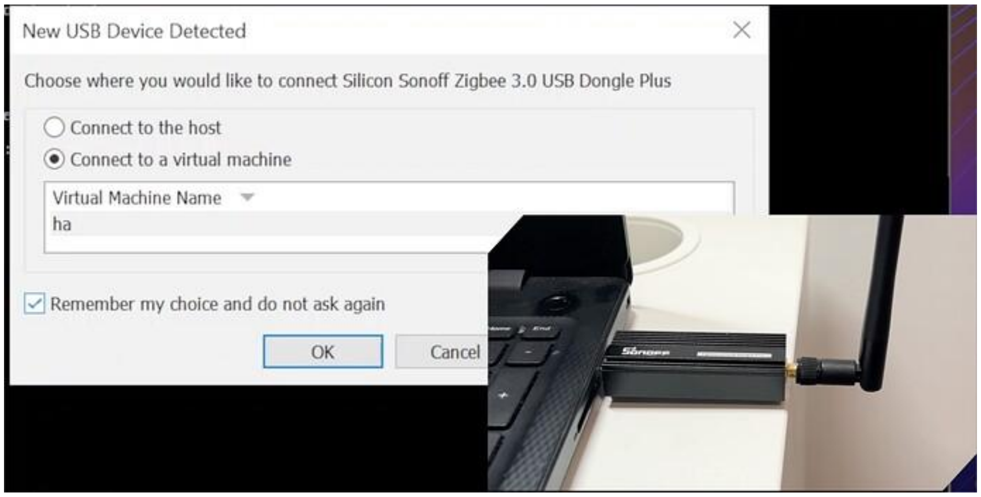
```
Waiting for the Home Assistant CLI to be ready...

Welcome to the Home Assistant command line.

Waiting for Supervisor to startup...
System information
  IPv4 addresses for enp2s1: 192.168.31.17/2
  IPv6 addresses for enp2s1: fdd0:4c80:cb91:4233:d58b:e182:9c64:db9c/64, fe80::2e29:6061:64bb:1f63/64

  OS Version:               Home Assistant OS 10.0
  Home Assistant Core:      landingpage

  Home Assistant URL:       http://homeassistant.local:8123
  Observer URL:             http://homeassistant.local:4357
ha >
```

Onboarding Home Assistant

Once completed, you will be able to reach Home Assistant on homeassistant.local:8123. If you are running an older Windows version or have a stricter network configuration, you might need to access Home Assistant at homeassistant:8123 or http://X.X.X.X:8123 (replace X.X.X.X with your's IP address as shown for IPv4 Addresses in the image above).

How To Add Zigbee Support

You can just connect your zigbee stick to your PC, and it should be detected like so:


New USB Device Detected

Choose where you would like to connect Silicon Sonoff Zigbee 3.0 USB Dongle Plus

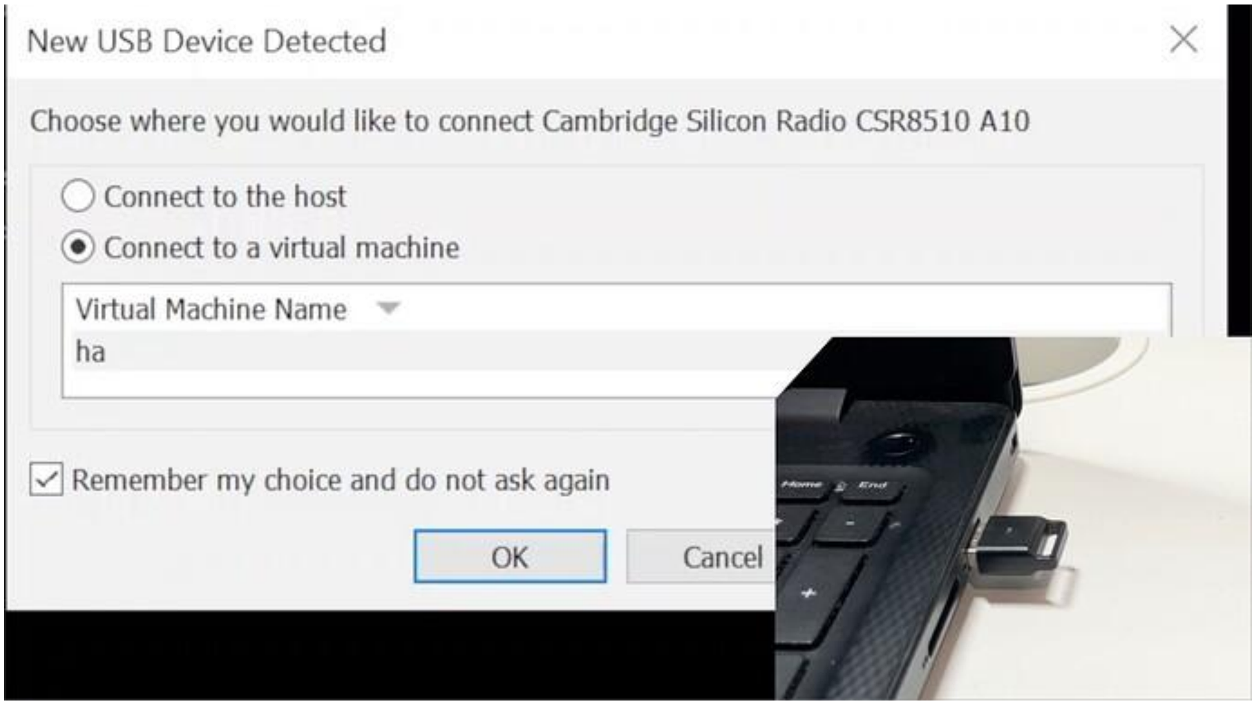
Connect to the host [✓ Connect to a virtual machine]

Virtual Machine Name: ha

[✓] Remember my choice and do not ask again

How To Add Bluetooth Support

You can just connect your Bluetooth dongle to your PC, and it should be detected like so:


New USB Device Detected

Choose where you would like to connect Cambridge Silicon Radio CSR8510 A10

Connect to the host [✓ Connect to a virtual machine]

Virtual Machine Name: ha

[✓] Remember my choice and do not ask again

That’s it! You have successfully installed Home Assistant on Windows using VMWare.

Here’s a simplified version of your instructions:

3.6.1 Install Home Assistant on VMware Workstation

1. Create a New Virtual Machine

1. Open **VMware Workstation** and select **Create a New Virtual Machine**.
2. Choose **I will install the operating system later**.
3. Select **Linux > Other Linux 5.x kernel 64-bit**.
4. Name the VM **home-assistant** and choose a storage location (e.g., C:\home-assistant).
5. Set the disk size and select **Store virtual disk as a single file**.
6. Click **Customize Hardware**:
   - Allocate **memory** and **CPU cores**.
   - Remove the **CD/DVD** device.
   - Under **Network Adapter**, choose **Bridged: Connected directly to the physical network**.
   - Click **Configure Adapters** and deselect all virtual adapters & Bluetooth devices.
   - Select your **host network adapter** (Ethernet).
7. Click **Finish** to complete the setup.

2. Replace the Disk File

1. Open **Windows Explorer** and go to C:\home-assistant.
2. Delete the **home-assistant.vmdk** file.
3. From your **Downloads** folder, find and unzip the **haos_ova_xx.x.vmdk** file.
4. Rename the extracted .vmdk file to **home-assistant.vmdk**.
5. Move the renamed file into C:\home-assistant.

3. Modify VM Settings

1. Right-click the **.vmx** file in C:\home-assistant and open it with **Notepad**.
2. Add the following line under .encoding:
3. firmware = "efi"

4. Start the Virtual Machine

1. Launch the **VM** and observe the boot process.
2. Once booted, access **Home Assistant** via:
    - homeassistant.local:8123
    - OR http://X.X.X.X:8123 (Replace X.X.X.X with your VM's IP address).

If you encounter a **.vmdk file not found** error, ensure you moved the correct file, not the entire folder.

2. Install the **ESPHome Add-on** in Home Assistant.
3. Add the ESP32 device to Home Assistant.
4. Monitor sensor data and control relays via the **Home Assistant dashboard**.

3.7 Alexa Integration

Alexa allows **voice control** of appliances using **relay-based automation**.

The Alexa integration allows users to control the Home Assistant entities via the Home Assistant Smart Home skill for Amazon Alexa. You can then say things like "Alexa, turn on the kitchen light" to control your local Home Assistant.

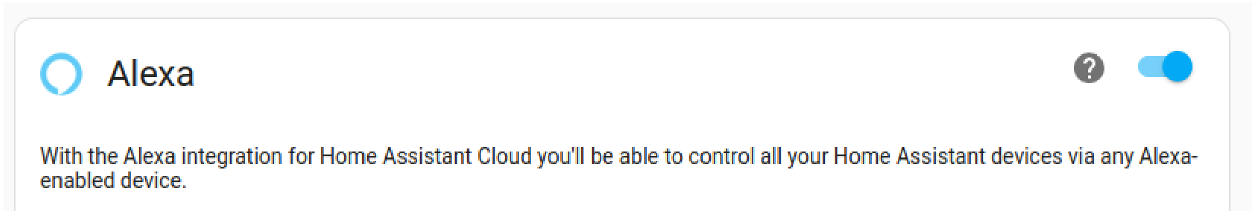
To use this integration, you need to have an Amazon Alexa enabled device like the Amazon Echo.

Note: The procedure below applies to Home Assistant version 2023.5 and later. If you are running an older version, Amazon Alexa is located under **Home Assistant Cloud**.

To control Home Assistant entities from Alexa, follow these steps:

1. Set up Home Assistant Cloud.

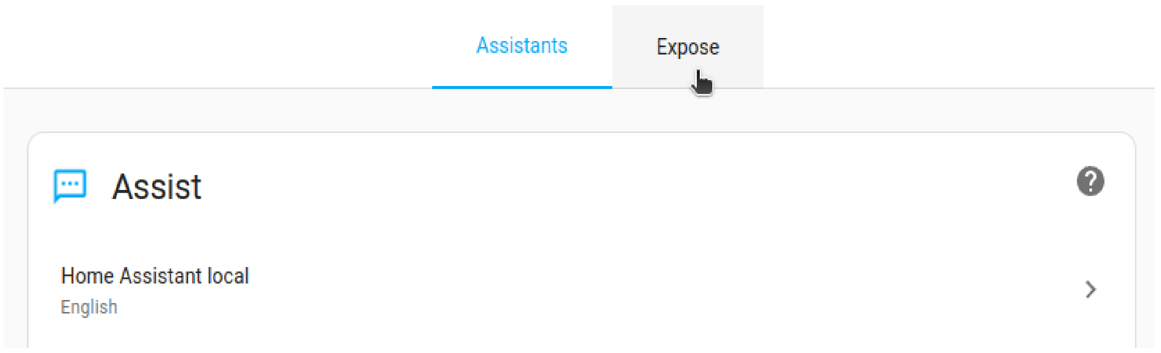
2. Under **Settings > Voiceassistant**,enable **Alexa**.



With the Alexa integration for Home Assistant Cloud you'll be able to control all your Home Assistant devices via any Alexa-enabled device.

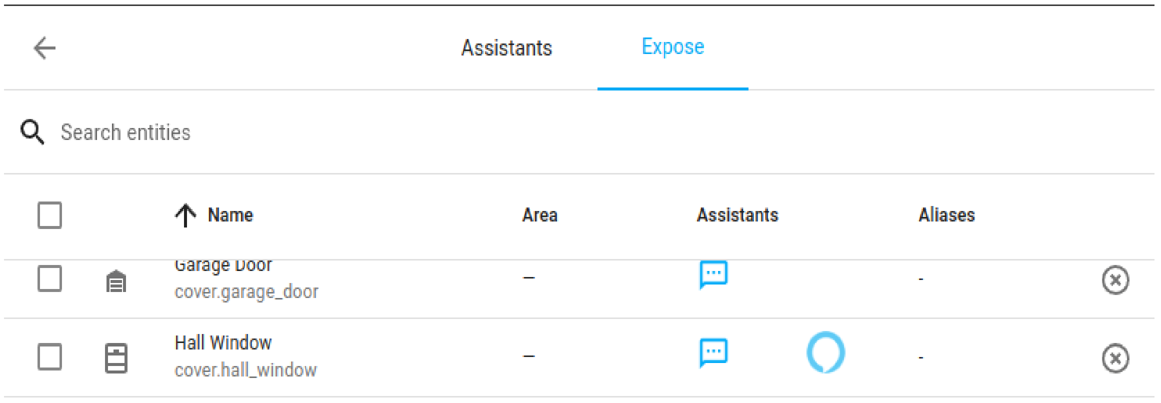
3. On the **Expose** tab, expose at least one entity to Alexa.
   - Open

     the **Expose** tab.



   - Select all entities you want to be able to control via Alexa.
   - Select **Expose enitites**. **Result**: The Alexa icon is now visible in

     the **Assistants** column.



		Name	Area	Assistants		Aliases	
☐		Garage Door cover.garage_door	–	💬		-	⊗
☐		Hall Window cover.hall_window	–	💬	○	-	⊗

4. Activate the Home Assistant Smart Home skill via the Alexa app.
   - From the Alexa App on your phone, go to **Skills & Games**.
   - Find **Home Assistant** and add it.

   The Home Assistant Smart Home skill is available in multiple marketplaces**Result**: You will be prompted to link to your Nabu Casa account.

5. Once activated, tell Alexa to discover new devices: *Alexa, discover new devices*.

3.7.1 Manual configuration

You can use configuration.yaml to configure the entities that are being shown to Alexa and how they are exposed.

If you use any filters, as shown in the example below, the entities can no longer be exposed via User Interface. They are still listed under **Settings > Voice assistant** > **Expose**, but are shown grayed out.

```
# Example configuration.yaml entry configuring Alexa


cloud:
  alexa:
    filter:
      include_entities:
        - alarm_control_panel.house
        - light.kitchen
        - light.kitchen_left
      include_domains:
        - switch
        - alarm_control_panel
      exclude_entities:
        - switch.outside
    entity_config:
      light.kitchen:
        name: Custom Name for Alexa
        description: The light in the kitchen
      switch.stairs:
        display_categories: LIGHT
```

Configuration variables

alexa:

  (map) (Optional) Configuration options for the Amazon Alexa integration.

  filter:

    (map) (Optional) description: Filters for entities to include/exclude from Alexa.

    include_entities:

      (list) (Optional) description: Entity IDs to include.

    include_domains:

      (list) (Optional) Domains to include.

    exclude_entities:

      (list) (Optional) Entity IDs to exclude.

    exclude_domains:

      (list) (Optional) Domains to exclude.

  entity_config:

    (map) (Optional) Entity specific configuration for Alexa.

    ENTITY_ID:

      (map) (Optional) Entity to configure.

      name:

        (string) (Optional) Name of entity to show in Alexa.

      description:

        (string) (Optional) Description of entity to show in Alexa.

      display_categories:

        (string) (Optional) The display category to use in Alexa.

- The integration requires **Amazon Alexa** and **Home Assistant Cloud** or **Emulated Hue** to recognize ESP32 as a smart device.
- Alexa **receives voice commands** like:
    - "Alexa, turn on the AC" → Activates relay connected to the AC.
    - "Alexa, turn off the fan" → Deactivates the fan relay.
- **Home Assistant automations** are configured to link Alexa commands with ESPHome actions.

3.8 Network and Connectivity Setup

❖ Network Overview

Component	Details
Home Assistant OS	Running on VMware
Internet Source	Mobile Hotspot
ESP32 Connectivity	Connected to Home Assistant
Connection Type	Wi-Fi (Hotspot)
IP Assignment	Dynamic (via Mobile Hotspot DHCP)
Remote Access	Not Configured

❖ Network Configuration Details

A. Home Assistant OS (VMware)

- **Virtual Network Mode:** Bridged / NAT
- **Assigned IP Address:** *(Check via ip a or ifconfig in terminal)*
- **Wi-Fi Connection Method:** Configured via VMware network settings

- **Gateway (Router IP):** *(Check from mobile hotspot settings)*
- **DNS Server:** 8.8.8.8 (Google DNS)

B. ESP32 Connectivity

- **Wi-Fi Mode:** Station Mode (STA)
- **Connected to Hotspot: Yes**
- **Assigned IP:** *(Check using WiFi.localIP(); in ESP32 code)*
- **MQTT/Web API Integration:** *(If applicable, mention protocol used)*

❖ Network Performance Analysis

Test	Result
Ping to Home Assistant (ping <HA IP>)	*(Check response time in ms)*
Ping to ESP32 (ping <ESP32 IP>)	*(Check response time in ms)*
Internet Latency (Google Test) (ping 8.8.8.8)	*(Avg response time in ms)*
Network Stability	*(Good / Moderate / Unstable - Based on test results)*

❖ Identified Issues & Recommendations

Issue	**Recommendation**
Dynamic IP Changes (Due to mobile hotspot)	Assign a **Static IP** to Home Assistant and ESP32 using DHCP reservation in hotspot settings.
High Latency Issues (If detected)	Use **2.4GHz Wi-Fi only** (Disable 5GHz in hotspot settings). Keep device close to the hotspot.

Issue	Recommendation
Remote Access Not Available	Consider **Tailscale VPN** or **DuckDNS + Let's Encrypt** for secure external access.
ESP32 Disconnections	Check **Wi-Fi signal strength** (WiFi.RSSI(); in ESP32 code). Keep ESP32 near the hotspot.

3.8.1 Conclusion & Next Steps

- **Verify IP addresses** of Home Assistant and ESP32.
- **Test connectivity** using ping and MQTT/Web requests.
- **Optimize network settings** in mobile hotspot (static IP, DNS settings).
- **Set up remote access** if needed for external control.

3.8.2 Protocols Used in Home Automation OS and Nodes:

1. Wi-Fi (IEEE 802.11) – Main Communication Protocol

- **Used for:** Connecting ESP32 to **Home Assistant**, **Alexa**, and the **mobile app**.
- **Why?**
    - Allows **ESP32 to communicate wirelessly** with Home Assistant.
    - Enables remote control of relays and sensors.

You are using Wi-Fi for ESP32 to connect with Home Assistant.

2. ESPHome API – Direct Communication with Home Assistant

- **Used for:** Sending sensor data and receiving relay control commands.
- **Why?**
    - **No need for an external MQTT broker** (simpler setup).
    - **Real-time data updates** between ESP32 and Home Assistant.
    - **Faster and more stable** than MQTT for small projects.

You are using ESPHome API for ESP32 to Home Assistant communication.

3. HTTP API – Communication Between Alexa and Home Assistant

- **Used for:** Controlling relays via Alexa voice commands.
- **Why?**
    - Alexa sends **HTTP requests** to Home Assistant to turn relays **ON/OFF**.
    - Used for **voice control integration**.

You are using HTTP API for Alexa to communicate with Home Assistant.

4. GPIO (General-Purpose Input/Output) – Direct Control of Sensors & Relays

- **Used for:** Connecting **DHT11, relays, and other sensors** to ESP32.
- **Why?**
    - **DHT11 sends temperature data** via GPIO pins.
    - **Relays receive ON/OFF signals** through GPIO.

You are using GPIO for ESP32 to control relays and read sensors

5.Protocols Used in This Project:

Communication Type	**Protocol Used**	**Used For**
ESP32 ↔ Home Assistant	**ESPHome API**	Sensor data, relay control
ESP32 ↔ Wi-Fi	**Wi-Fi (802.11)**	Internet connectivity
Alexa ↔ Home Assistant	**HTTP API**	Alexa voice control
Sensors ↔ ESP32	**GPIO**	Reading temperature (DHT11)
Relays ↔ ESP32	**GPIO**	Turning appliances ON/OFF

Pin Configuration

4.1 ESP32 GPIO Pin Explanation

- **ESP32 Peripherals**

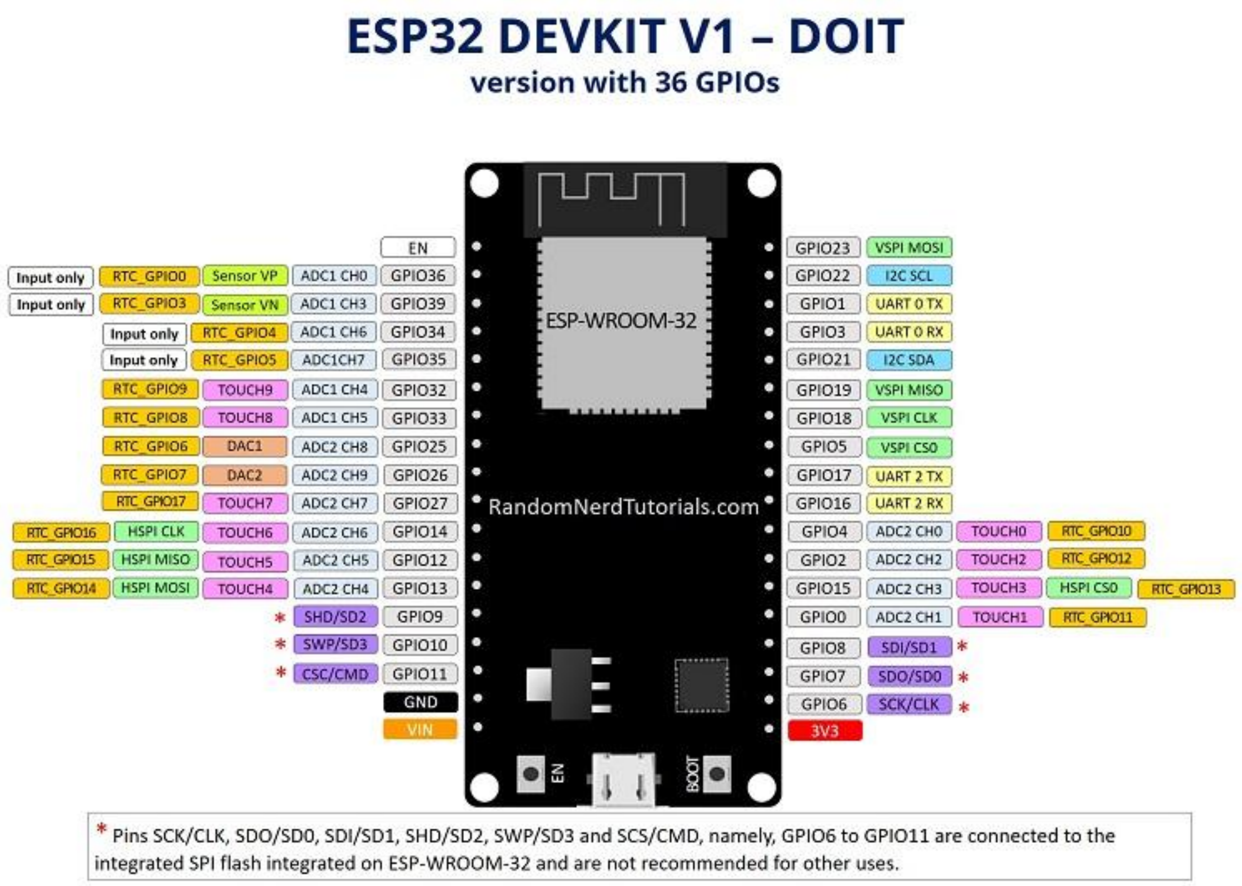
  - 18 Analog-to-Digital Converter (ADC) channels
  - 3 SPI interfaces
  - 3 UART interfaces
  - 2 I2C interfaces
  - 16 PWM output channels
  - 2 Digital-to-Analog Converters (DAC)
  - 2 I2S interfaces
  - 10 Capacitive sensing GPIOs

The ADC (analog to digital converter) and DAC (digital to analog converter) features are assigned to specific static pins. However, you can decide which pins are UART, I2C, SPI, PWM, etc – you just need to assign them in the code. This is possible due to the ESP32 chip's multiplexing feature.

Although you can define the pins properties on the software, there are pins assigned by default as shown in the following figure (this is an example for the ESP32 DEVKIT V1 DOIT board with 36 pins – the pin location can change depending on the manufacturer).

Additionally, there are pins with specific features that make them suitable or not for a particular project. The following table shows what pins are best to use as inputs, outputs and which ones you need to be cautious.

The pins highlighted in green are OK to use. The ones highlighted in yellow are OK to use, but you need to pay attention because they may have an unexpected behavior mainly at boot. The pins highlighted in red are not recommended to use as inputs or outputs.


ESP32 DEVKIT V1 – DOIT
version with 36 GPIOs
EN
Input only RTC_GPIO0 Sensor VP ADC1 CH0 GPIO36
Input only RTC_GPIO3 Sensor VN ADC1 CH3 GPIO39
Input only RTC_GPIO4 ADC1 CH6 GPIO34
Input only RTC_GPIO5 ADC1CH7 GPIO35
RTC_GPIO9 TOUCH9 ADC1 CH4 GPIO32
RTC_GPIO8 TOUCH8 ADC1 CH5 GPIO33
RTC_GPIO6 DAC1 ADC2 CH8 GPIO25
RTC_GPIO7 DAC2 ADC2 CH9 GPIO26
RTC_GPIO17 TOUCH7 ADC2 CH7 GPIO27
RTC_GPIO16 HSPI CLK TOUCH6 ADC2 CH6 GPIO14
RTC_GPIO15 HSPI MISO TOUCH5 ADC2 CH5 GPIO12
RTC_GPIO14 HSPI MOSI TOUCH4 ADC2 CH4 GPIO13
* SHD/SD2 GPIO9
* SWP/SD3 GPIO10
* CSC/CMD GPIO11
GND
VIN
GPIO23 VSPI MOSI
GPIO22 I2C SCL
GPIO1 UART 0 TX
GPIO3 UART 0 RX
GPIO21 I2C SDA
GPIO19 VSPI MISO
GPIO18 VSPI CLK
GPIO5 VSPI CS0
GPIO17 UART 2 TX
GPIO16 UART 2 RX
GPIO4 ADC2 CH0 TOUCH0 RTC_GPIO10
GPIO2 ADC2 CH2 TOUCH2 RTC_GPIO12
GPIO15 ADC2 CH3 TOUCH3 HSPI CS0 RTC_GPIO13
GPIO0 ADC2 CH1 TOUCH1 RTC_GPIO11
GPIO8 SDI/SD1 *
GPIO7 SDO/SD0 *
GPIO6 SCK/CLK *
3V3
ESP-WROOM-32
RandomNerdTutorials.com
EN
BOOT

* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

GPIO	Input	Output	Notes
0	pulled up	OK	outputs PWM signal at boot, must be LOW to enter flashing mode
1	TX pin	OK	debug output at boot
2	OK	OK	connected to on-board LED, must be left floating or LOW to enter flashing mode
3	OK	RX pin	HIGH at boot

4	OK	OK	
5	OK	OK	outputs PWM signal at boot, strapping pin
6	X	x	connected to the integrated SPI flash
7	X	x	connected to the integrated SPI flash
8	X	x	connected to the integrated SPI flash
9	X	x	connected to the integrated SPI flash
10	X	x	connected to the integrated SPI flash
11	X	x	connected to the integrated SPI flash
12	OK	OK	boot fails if pulled high, strapping pin
13	OK	OK	
14	OK	OK	outputs PWM signal at boot
15	OK	OK	outputs PWM signal at boot, strapping pin
16	OK	OK	
17	OK	OK	
18	OK	OK	
19	OK	OK	
21	OK	OK	
22	OK	OK	

23	OK	OK	
25	OK	OK	
26	OK	OK	
27	OK	OK	
32	OK	OK	
33	OK	OK	
34	OK		input only
35	OK		input only
36	OK		input only
39	OK		input only

analysis of the ESP32 GPIOs and its functions.

❖ Input only pins

GPIOs 34 to 39 are GPIs – input only pins. These pins don't have internal pull-up or pull-down resistors. They can't be used as outputs, so use these pins only as inputs:

- GPIO 34
- GPIO 35
- GPIO 36
- GPIO 39

❖ SPI flash integrated on the ESP-WROOM-32

GPIO 6 to GPIO 11 are exposed in some ESP32 development boards. However, these pins are connected to the integrated SPI flash on the ESP-WROOM-32 chip and are not recommended for other uses. So, don’t use these pins in your projects:

- GPIO 6 (SCK/CLK)
- GPIO 7 (SDO/SD0)
- GPIO 8 (SDI/SD1)
- GPIO 9 (SHD/SD2)
- GPIO 10 (SWP/SD3)
- GPIO 11 (CSC/CMD)

❖ Capacitive touch GPIOs

The ESP32 has 10 internal capacitive touch sensors. These can sense variations in anything that holds an electrical charge, like the human skin. So they can detect variations induced when touching the GPIOs with a finger. These pins can be easily integrated into capacitive pads and replace mechanical buttons. The capacitive touch pins can also be used to wake up the ESP32 from deep sleep.
Those internal touch sensors are connected to these GPIOs:

- T0 (GPIO 4)
- T1 (GPIO 0)
- T2 (GPIO 2)
- T3 (GPIO 15)
- T4 (GPIO 13)
- T5 (GPIO 12)
- T6 (GPIO 14)
- T7 (GPIO 27)
- T8 (GPIO 33)
- T9 (GPIO 32)

Learn how to use the touch pins with Arduino IDE: ESP32 Touch Pins with Arduino IDE

❖ Analog to Digital Converter (ADC)

The ESP32 has 18 x 12 bits ADC input channels (while the ESP8266 only has 1x 10 bits ADC). These are the GPIOs that can be used as ADC and respective channels:

- ADC1_CH0 (GPIO 36)
- ADC1_CH1 (GPIO 37)
- ADC1_CH2 (GPIO 38)
- ADC1_CH3 (GPIO 39)
- ADC1_CH4 (GPIO 32)
- ADC1_CH5 (GPIO 33)
- ADC1_CH6 (GPIO 34)
- ADC1_CH7 (GPIO 35)
- ADC2_CH0 (GPIO 4)
- ADC2_CH1 (GPIO 0)
- ADC2_CH2 (GPIO 2)
- ADC2_CH3 (GPIO 15)
- ADC2_CH4 (GPIO 13)
- ADC2_CH5 (GPIO 12)
- ADC2_CH6 (GPIO 14)
- ADC2_CH7 (GPIO 27)
- ADC2_CH8 (GPIO 25)
- ADC2_CH9 (GPIO 26)

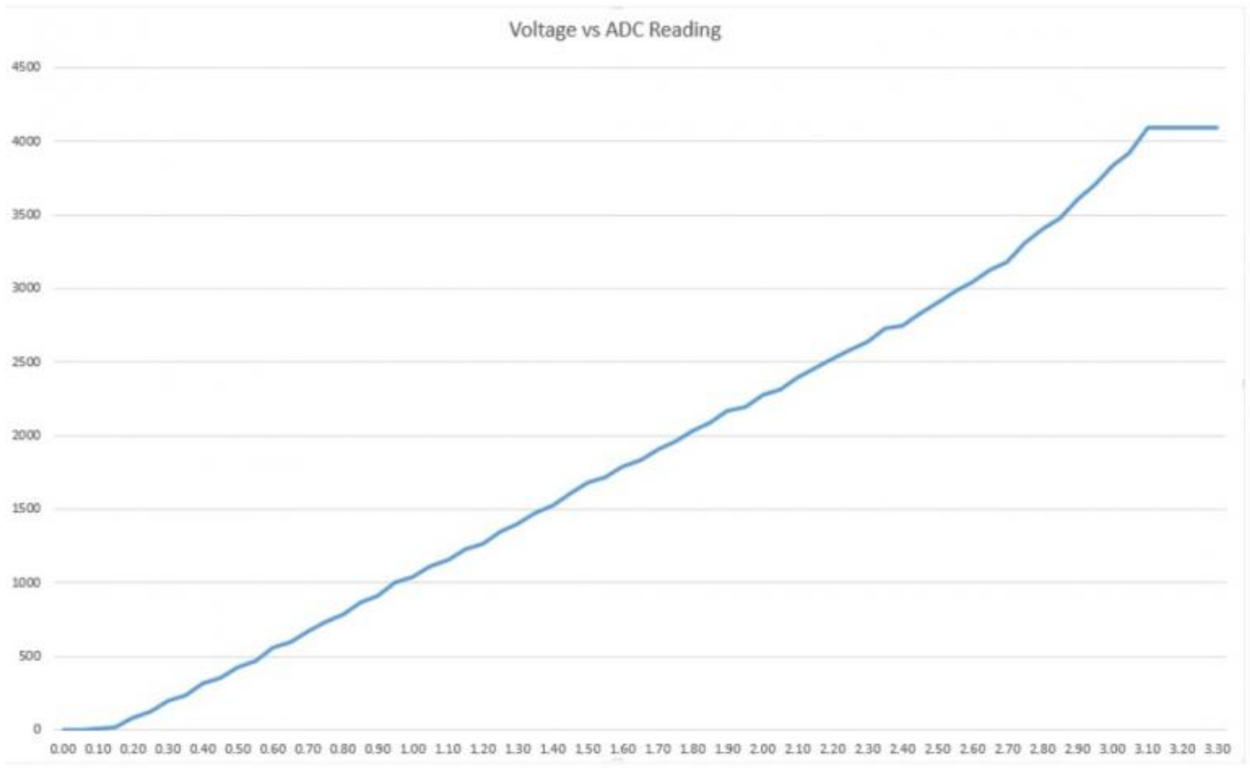
❖ Learn how to use the ESP32 ADC pins:

- ESP32 ADC Pins with Arduino IDE
- ESP32 ADC Pins with MicroPython
-

Note: ADC2 pins cannot be used when Wi-Fi is used. So, if you’re using Wi-Fi and you’re having trouble getting the value from an ADC2 GPIO, you may consider using an ADC1 GPIO instead. That should solve your problem.

The ADC input channels have a 12-bit resolution. This means that you can get analog readings ranging from 0 to 4095, in which 0 corresponds to 0V and 4095 to 3.3V. You can also set the resolution of your channels on the code and the ADC range.

The ESP32 ADC pins don't have a linear behavior. You'll probably won't be able to distinguish between 0 and 0.1V, or between 3.2 and 3.3V. You need to keep that in mind when using the ADC pins. You'll get a behavior similar to the one shown in the following figure.



❖ Digital to Analog Converter (DAC)

There are 2 x 8 bits DAC channels on the ESP32 to convert digital signals into analog voltage signal outputs. These are the DAC channels:

- DAC1 (GPIO25)
- DAC2 (GPIO26)

❖ RTC GPIOs

There is RTC GPIO support on the ESP32. The GPIOs routed to the RTC low-power subsystem can be used when the ESP32 is in deep sleep. These RTC GPIOs can be used to wake up the ESP32 from deep sleep when the Ultra Low Power (ULP) co-processor is running. The following GPIOs can be used as an external wake up source.

- RTC_GPIO0 (GPIO36)
- RTC_GPIO3 (GPIO39)
- RTC_GPIO4 (GPIO34)
- RTC_GPIO5 (GPIO35)
- RTC_GPIO6 (GPIO25)
- RTC_GPIO7 (GPIO26)
- RTC_GPIO8 (GPIO33)
- RTC_GPIO9 (GPIO32)
- RTC_GPIO10 (GPIO4)
- RTC_GPIO11 (GPIO0)
- RTC_GPIO12 (GPIO2)
- RTC_GPIO13 (GPIO15)
- RTC_GPIO14 (GPIO13)
- RTC_GPIO15 (GPIO12)
- RTC_GPIO16 (GPIO14)
- RTC_GPIO17 (GPIO27)

Learn how to use the RTC GPIOs to wake up the ESP32 from deep sleep: ESP32 Deep Sleep with Arduino IDE and Wake Up Sources

❖ PWM

The ESP32 LED PWM controller has 16 independent channels that can be configured to generate PWM signals with different properties. All pins that can act as outputs can be used as PWM pins (GPIOs 34 to 39 can't generate PWM).

To set a PWM signal, you need to define these parameters in the code:

- Signal's frequency;
- Duty cycle;
- PWM channel;
- GPIO where you want to output the signal.

Learn how to use ESP32 PWM with Arduino IDE: ESP32 PWM with Arduino IDE

❖ I2C

The ESP32 has two I2C channels and any pin can be set as SDA or SCL. When using the ESP32 with the Arduino IDE, the default I2C pins are:

- GPIO 21 (SDA)
- GPIO 22 (SCL)

If you want to use other pins when using the wire library, you just need to call:

```
Wire.begin(SDA, SCL);
```

Learn more about I2C communication protocol with the ESP32 using Arduino IDE: ESP32 I2C Communication (Set Pins, Multiple Bus Interfaces and Peripherals)

More I2C Tutorials with the ESP32:

- ESP32 I2C Master and Slave (I2C Communication Between Two ESP32) – Arduino IDE
- ESP32: I2C Scanner (Arduino IDE) – Finding the Address of I2C Devices
- Guide for TCA9548A I2C Multiplexer with the ESP32

❖ SPI

By default, the pin mapping for SPI is:

SPI	MOSI	MISO	CLK	CS
VSPI	GPIO 23	GPIO 19	GPIO 18	GPIO 5
HSPI	GPIO 13	GPIO 12	GPIO 14	GPIO 15

Learn more about SPI communication protocol with the ESP32 using Arduino IDE: ESP32 SPI Communication: Set Pins, Multiple SPI Bus Interfaces, and Peripherals (Arduino IDE)

❖ UART

The ESP32 supports up to three UART interfaces: **UART0**, **UART1**, and **UART2**, depending on the ESP32 board model you're using.

- **UART0** is usually reserved for communication with the serial monitor during upload and debugging. However, you can also use it for communication with other devices after uploading the code if the Serial Monitor is not needed.
- **UART1** and **UART2**: available to communicate with external devices.

Like I2C and SPI, these UART pins can be mapped to any GPIO pin on the ESP32. However, they have a default pin assignment on most board models.

For most ESP32 boards the UART pin assignment is as follows:

UART Port	TX	RX	Remarks
UART0	GPIO 1	GPIO 3	Used for Serial Monitor and uploading code; Can be assigned to other GPIOs;

UART1	GPIO 10	GPIO 9	Must be assigned to other GPIOs
UART2	GPIO 17	GPIO 16	Can be assigned to other GPIOs

About UART1 (GPIO 9 and GPIO10) – these GPIOs are connected to the ESP32 SPI flash memory, so you can't use them like that. To use UART1 to communicate with other devices, you must define different pins using the HardwareSerial library.

Learn more about UART with the ESP32: ESP32 UART Communication (Serial): Set Pins, Interfaces, Send and Receive Data (Arduino IDE)

If you're using an ESP32-S3, the assignment is completely different. Check out the ESP32-S3 pinout here.

❖ Interrupts

All GPIOs can be configured as interrupts.

Learn how to use interrupts with the ESP32:

- ESP32 interrupts with Arduino IDE
- ESP32 interrupts with MicroPython

❖ Strapping Pins

The ESP32 chip has the following strapping pins:

- GPIO 0 (must be LOW to enter boot mode)
- GPIO 2 (must be floating or LOW during boot)
- GPIO 4
- GPIO 5 (must be HIGH during boot)
- GPIO 12 (must be LOW during boot)

- GPIO 15 (must be HIGH during boot)

These are used to put the ESP32 into bootloader or flashing mode. On most development boards with built-in USB/Serial, you don’t need to worry about the state of these pins. The board puts the pins in the right state for flashing or boot mode. More information on the ESP32 Boot Mode Selection can be found here.

However, if you have peripherals connected to those pins, you may have trouble trying to upload new code, flashing the ESP32 with new firmware, or resetting the board. If you have some peripherals connected to the strapping pins and you are getting trouble uploading code or flashing the ESP32, it may be because those peripherals are preventing the ESP32 from entering the right mode. Read the Boot Mode Selection documentation to guide you in the right direction. After resetting, flashing, or booting, those pins work as expected.

❖ Pins HIGH at Boot

Some GPIOs change their state to HIGH or output PWM signals at boot or reset. This means that if you have outputs connected to these GPIOs you may get unexpected results when the ESP32 resets or boots.

- GPIO 1
- GPIO 3
- GPIO 5
- GPIO 6 to GPIO 11 (connected to the ESP32 integrated SPI flash memory – not recommended to use).
- GPIO 14
- GPIO 15

❖ Enable (EN)

Enable (EN) is the 3.3V regulator’s enable pin. It’s pulled up, so connect to ground to disable the 3.3V regulator. This means that you can use this pin connected to a pushbutton to restart your ESP32, for example.

5.2 Pin Mapping Table

The **Pin Mapping Table** provides an overview of how different components (DHT11, relays, power supply, etc.) are connected to the **ESP32**.

❖ **ESP32 Pin Configuration for Your Home Automation Project**

Component	ESP32 Pin	Purpose
DHT11 Sensor	GPIO4	Reads temperature and humidity
Relay 1 (Fan/Light)	GPIO26	Controls appliance 1
Relay 2 (AC Control)	GPIO27	Controls appliance 2
Relay 3	GPIO14	Controls appliance 3
Relay 4	GPIO12	Controls appliance 4
Wi-Fi Module	Built-in	Wireless communication
Power Supply Input	VIN (5V)	Powering ESP32
GND (Ground)	GND	Common ground for all components

5.3 DHT11 Connection with ESP32

The **DHT11** sensor measures **temperature and humidity** and sends data to **ESP32** through a single digital pin.

❖ **Connections**

DHT11 Pin	ESP32 Pin	Description
VCC	3.3V or 5V	Power supply
GND	GND	Ground
Data	GPIO4	Temperature and humidity data

You can use ESPHome to read data from GPIO4.

5.3.1 How It Works:

1. The **ESP32 sends a signal** to the **DHT11 sensor** to start reading.
2. The **DHT11 measures temperature and humidity** and sends the data back as a digital signal.
3. **ESP32 processes the data** and sends it to **Home Assistant**.
4. If temperature **exceeds a set threshold (e.g., 28°C)**, the **AC relay is turned ON** automatically.

5.3.2 Code Example in ESPHome (YAML Format):

```
yaml
sensor:
  - platform: dht
    pin: GPIO4
    model: DHT11
    temperature:
      name: "Room Temperature"
    humidity:
      name: "Room Humidity"
    update_interval: 10s
```

4.4 Relay Connection with ESP32

The **4-Channel Relay Module** is used to **control home appliances** like **lights, fans, and AC** using **ESP32**.

❖ Connections

Relay Module Pin	**ESP32 Pin**	**Description**
VCC	5V	Power supply
GND	GND	Ground
IN1	GPIO26	Controls relay 1 (e.g., Fan)
IN2	GPIO27	Controls relay 2 (e.g., AC)
IN3	GPIO14	Controls relay 3
IN4	GPIO12	Controls relay 4

Each relay is connected to an ESP32 GPIO pin to control different appliances.

4.4.1 How It Works:

1. **ESP32 sends HIGH (3.3V) or LOW (0V) signals** to control the relays.
2. **When ESP32 sends a HIGH signal**, the **relay turns ON** (appliance powered).
3. **When ESP32 sends a LOW signal**, the **relay turns OFF** (appliance disconnected).
4. The **Home Assistant dashboard** and **Alexa commands** control the relays remotely.

❖ ESPHome Code for Controlling Relays (YAML Format):

yaml

CopyEdit

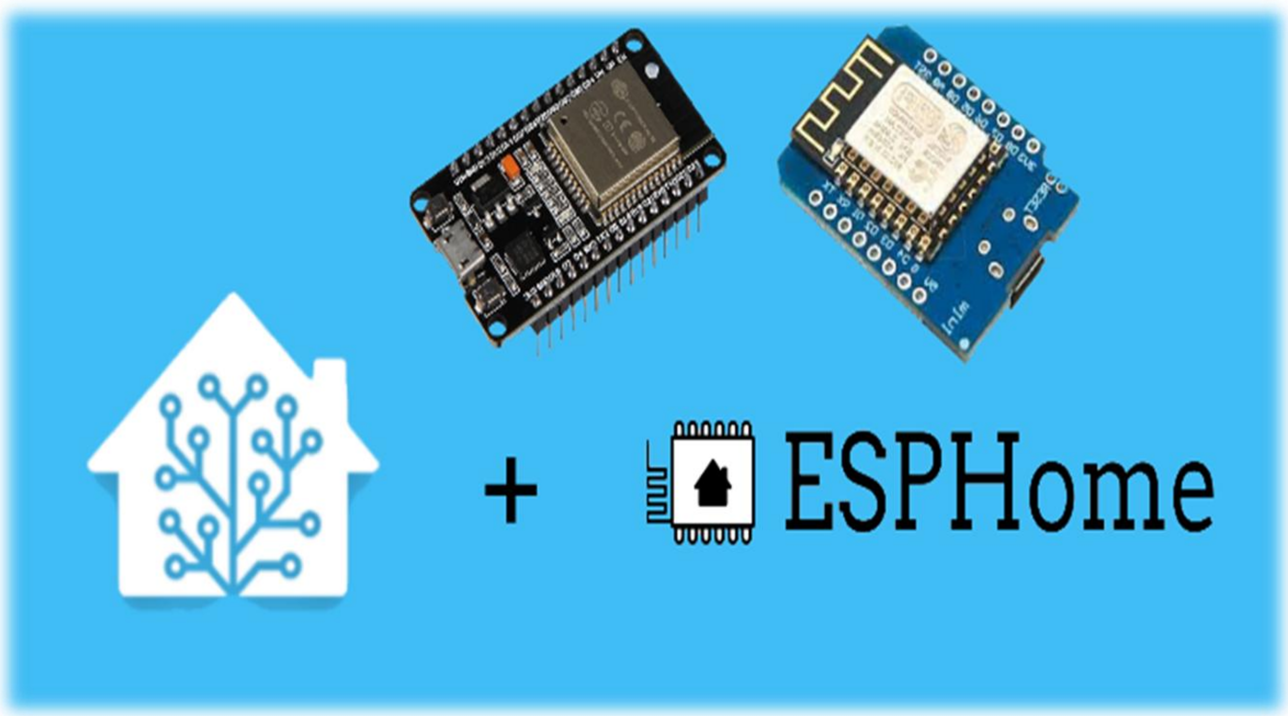
```
switch:
  - platform: gpio
```

```
   pin: GPIO26
   name: "Fan Relay"
   inverted: false
 - platform: gpio
   pin: GPIO27
   name: "AC Relay"
   inverted: false
 - platform: gpio
   pin: GPIO14
   name: "Light Relay"
   inverted: false
 - platform: gpio
   pin: GPIO12
   name: "Other Relay"
   inverted: false
```

5.4.2 Final Working Flow

- **DHT11 reads the temperature** and sends data to ESP32.
- **ESP32 processes the data** and determines if AC should be turned on.
- **If temperature > 28°C**, ESP32 **activates the AC relay** (GPIO27 → HIGH).
- **User can control relays manually** via **Home Assistant UI** or **Alexa voice commands**.
- The **system updates Home Assistant** with the latest temperature and relay status.

ESPHome Configuration



ESPHome Configuration for Home Automation System

5.1 Introduction to ESPHome

ESPHome is an open-source firmware used for configuring ESP32 and ESP8266 microcontrollers with YAML-based configurations. It simplifies IoT development by eliminating the need for traditional programming, allowing seamless integration with Home Assistant for smart home automation.

5.1.1 Key Features

- **YAML-Based Configuration** – Easy setup without coding.
- **Home Assistant Integration** – Auto-detects devices in Home Assistant.
- **OTA Updates** – Allows wireless firmware updates.
- **Logging & Debugging** – Real-time monitoring of sensor data and logs.

5.2 Writing YAML Configuration

A YAML configuration file is used to define the settings for ESP32, including Wi-Fi credentials, sensor definitions, relay configurations, and API integration.

❖ Example YAML Configuration for ESP32

```
esphome:
  name: esp32-home-automation
  platform: ESP32
  board: esp32dev

wifi:
  ssid: "Your_WiFi_Name"
  password: "Your_WiFi_Password"
  manual_ip:
```

```yaml
    static_ip: 192.168.1.100
    gateway: 192.168.1.1
    subnet: 255.255.255.0

api:

logger:

ota:

sensor:
  - platform: dht
    pin: GPIO4
    model: DHT11
    temperature:
      name: "Room Temperature"
    humidity:
      name: "Room Humidity"
    update_interval: 10s

switch:
  - platform: gpio
    pin: GPIO26
    name: "Fan Relay"
  - platform: gpio
    pin: GPIO27
    name: "AC Relay"
  - platform: gpio
    pin: GPIO14
    name: "Light Relay"
  - platform: gpio
    pin: GPIO12
    name: "Other Relay"
```

5.3 Detailed Explanation of the YAML Code

Code Section	Purpose
esphome:	Defines the ESP32 device name and board type.
wifi:	Connects ESP32 to Wi-Fi with a static IP.
api:	Enables communication with Home Assistant.
logger:	Provides real-time debugging logs.
ota:	Enables Over-The-Air updates.
sensor:	Reads temperature and humidity data from the DHT11 sensor.
switch:	Controls relays for AC, fan, light, and other appliances.

5.4 Logging and Debugging ESPHome

❖ How to View Logs?

1. **Using ESPHome Dashboard:**
   - Open the ESPHome web dashboard.
   - Click on your ESP32 device.
   - Click **Logs** to see live sensor readings and errors.
2. **Using Command Line (USB Connection):**
3. esphome logs esp32-home-automation.yaml

❖ Common Debugging Issues & Fixes

Issue	Solution
ESP32 not connecting to Wi-Fi	Check SSID and password. Ensure static IP is correctly configured.
Sensor data not updating	Verify GPIO connections and update interval.
Relays not switching	Check correct GPIO pin assignments and power supply.

5.5 Uploading Code to ESP32

Method 1: Flashing via USB (First-Time Setup)

1. Connect ESP32 to a computer via USB.
2. Install ESPHome CLI:
3. pip install esphome
4. Flash firmware to ESP32:
5. esphome run esp32-home-automation.yaml

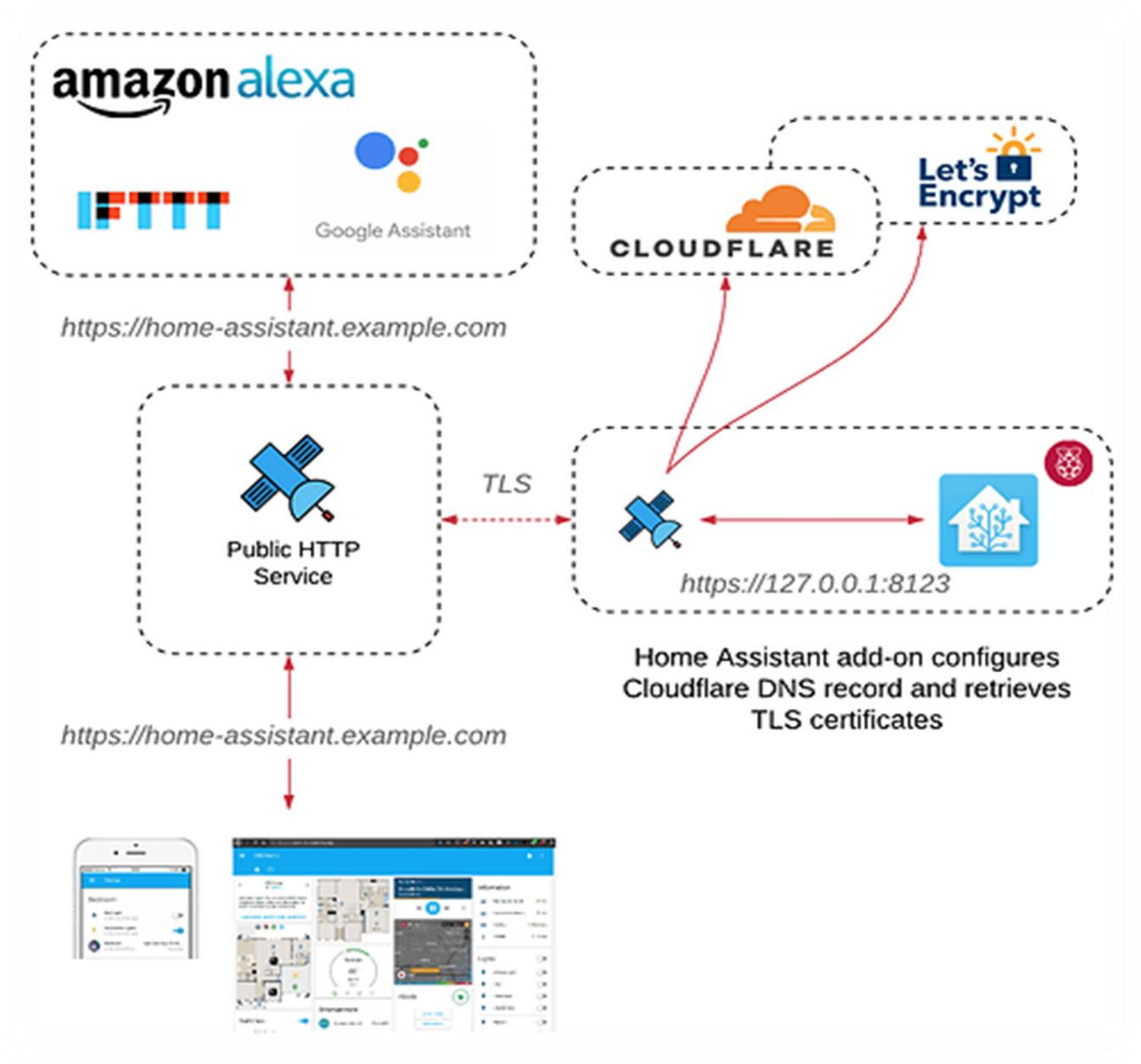
Method 2: OTA (Over-The-Air Updates)

Once the initial USB flashing is done, future updates can be done wirelessly:

esphome upload esp32-home-automation.yaml

ESP32 will receive the update over Wi-Fi without requiring a USB connection.

Working Principle


amazon alexa
IFTTT
Google Assistant
CLOUDFLARE
Let's Encrypt
https://home-assistant.example.com
TLS
Public HTTP Service
https://127.0.0.1:8123
Home Assistant add-on configures Cloudflare DNS record and retrieves TLS certificates
https://home-assistant.example.com

6.1 Step-by-Step Process of System Functionality

1. **System Initialization**: ESP32 boots up, initializing all sensors, relays, and network connections.
2. **Temperature Reading**: The DHT11 sensor reads the ambient temperature and humidity values.
3. **Data Processing**: ESP32 processes the temperature data and evaluates it against the set threshold.
4. **Decision Making**: Based on the temperature reading, ESP32 decides whether to turn the AC on or off.
5. **Relay Activation**: If the temperature exceeds the threshold, the ESP32 triggers the relay to turn on the AC; otherwise, it remains off.
6. **Home Assistant Update**: The ESP32 sends real-time data updates to Home Assistant for monitoring.
7. **User Interaction**: Users can manually control appliances through the Home Assistant dashboard or via Alexa voice commands.
8. **Alexa Execution**: When a user gives a voice command to Alexa, it sends a control signal to Home Assistant, which then triggers the ESP32 to perform the requested action.
9. **System Feedback**: Status updates are displayed on the Home Assistant UI, and logs are recorded for debugging and monitoring purposes.

6.2 How ESP32 Reads Temperature

1. The DHT11 sensor is connected to a GPIO pin of the ESP32.
2. ESP32 sends a request signal to the DHT11 sensor to obtain temperature and humidity values.
3. DHT11 responds by sending a digital signal containing temperature and humidity data.
4. ESP32 processes the received data and updates it in real-time.
5. The temperature values are displayed on the Home Assistant dashboard for user monitoring.

6.3 Decision Making for AC Control

1. **Temperature Comparison**: The ESP32 continuously compares the current temperature reading with a predefined threshold (e.g., 28°C).
2. **Conditional Operation**:
    - If the temperature exceeds 28°C, the ESP32 sends a signal to activate the relay, turning on the AC.
    - If the temperature drops below 28°C, the relay remains off, keeping the AC turned off.
3. **Data Logging**: The system logs temperature changes and relay status in Home Assistant.
4. **User Override**: Users can manually override the system through Home Assistant.

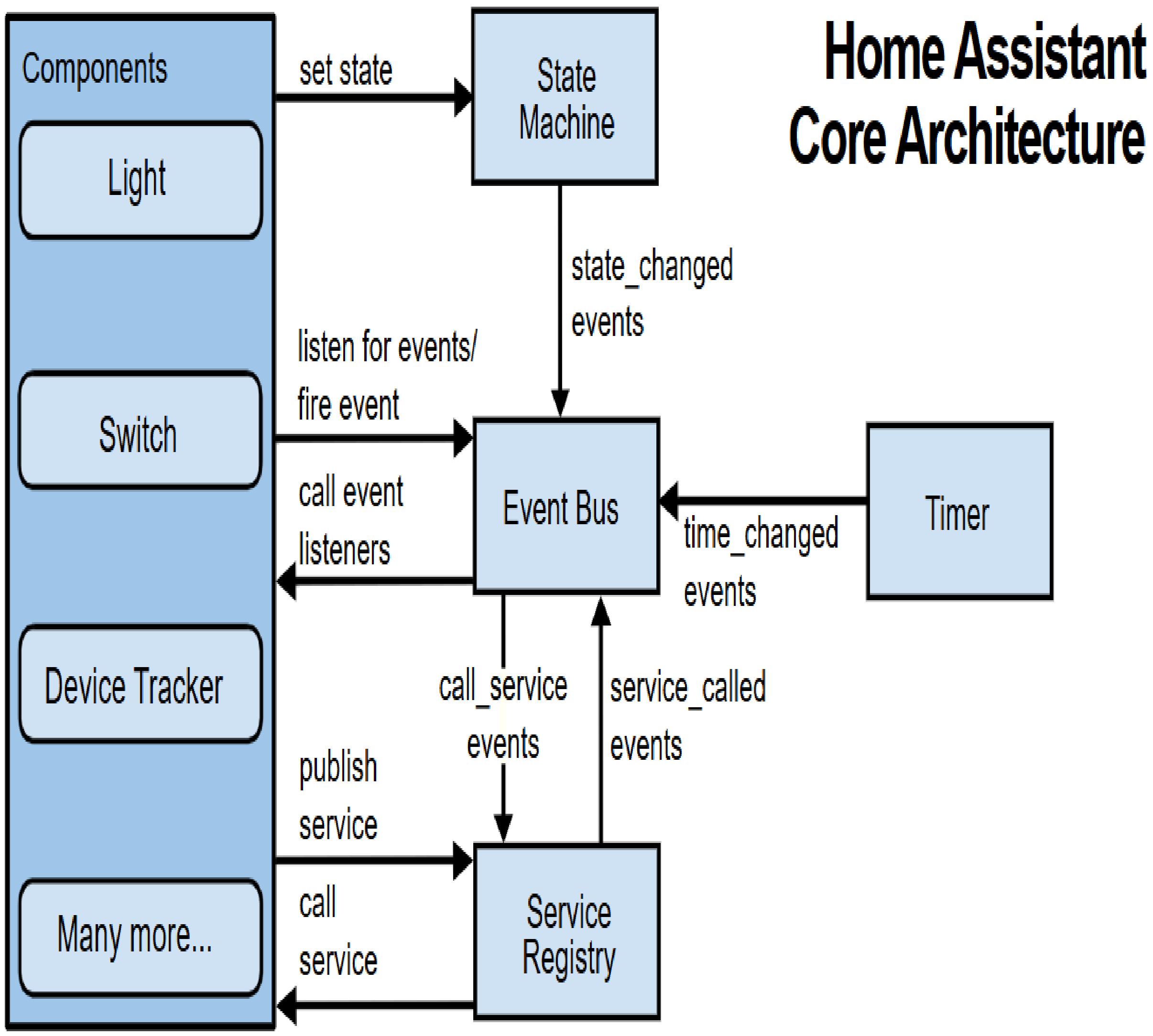
6.4 Manual Control via Home Assistant UI

1. Users access the Home Assistant dashboard via a web browser or mobile app.
2. The UI displays real-time sensor data and relay status.
3. Users can manually turn on/off relays to control appliances.
4. Commands sent via Home Assistant are transmitted to ESP32, triggering the corresponding relays.

6.5 Alexa Voice Command Execution

1. User issues a voice command (e.g., "Alexa, turn on the AC").
2. Alexa processes the voice command and sends a request to Home Assistant.
3. Home Assistant verifies the request and sends a control signal to ESP32.
4. ESP32 triggers the corresponding relay to execute the command.
5. The system provides feedback, confirming the successful execution of the action via Home Assistant UI.

Connectivity and Control



7.1 Home Assistant UI Overview

❖ What is Home Assistant?

Home Assistant is an open-source platform that enables smart home automation. It provides a user-friendly web interface to monitor and control IoT devices. The interface allows users to check real-time sensor data, control relays, and manage automation rules.

7.1.1 Features of Home Assistant UI in This Project:

1. **Real-time Data Monitoring:** View temperature readings from the DHT11 sensor.
2. **Device Control:** Manually switch relays on/off using the dashboard.
3. **Automation Setup:** Configure temperature-based AC automation.
4. **Alexa and Voice Control Integration:** Control devices using voice commands.

7.1.2 Home Assistant Dashboard Components:

- **Temperature Display:** Shows real-time readings from DHT11.
- **Relay Control Buttons:** Allows manual control of appliances.
- **Status Indicators:** Displays ON/OFF status of connected devices.
- **Automation Logs:** Records system activity for debugging.

7.2 Adding ESP32 to Home Assistant

To control ESP32 using Home Assistant, we need to integrate it via **ESPHome**. This involves the following steps:

Step 1: Install ESPHome in Home Assistant

- Navigate to **Home Assistant → Settings → Add-ons**
- Search for **ESPHome** and install it
- Start the ESPHome service

Step 2: Create a New ESPHome Device

- Open ESPHome and click on **New Device**
- Enter a **device name** (e.g., "ESP32_Home_Automation")
- Select **ESP32** as the device type
- Enter your **Wi-Fi credentials**

Step 3: Flash ESPHome Firmware to ESP32

- Connect ESP32 to your PC
- Use ESPHome **Web Flasher** or **command-line tool** to upload firmware
- Once flashed, ESP32 will automatically connect to Home Assistant

Step 4: Add ESP32 to Home Assistant

- In Home Assistant, go to **Devices & Services**
- Click on **ESPHome Integration → Add Device**
- Select the ESP32 from the list and click **Add**

Once added, you can monitor sensor readings and control relays from Home Assistant UI.

7.3 Controlling Relays from Home Assistant Dashboard

The relay module allows controlling home appliances using Home Assistant.

7.3.1 How the Relay Works in the Project:

- The **relay module** is connected to ESP32 GPIO pins
- When the relay is **ON**, it allows current to flow, turning the appliance ON
- When the relay is **OFF**, the circuit is disconnected, turning the appliance OFF

❖ **Steps to Control Relays in Home Assistant UI:**

1. **Go to Home Assistant Dashboard**
2. **Find the Relay Control Section**

3. **Click ON/OFF Toggle Buttons** to manually switch appliances
4. **Set Up Automation Rules** to automate relay switching based on temperature

Example:

- If **temperature > 28°C**, turn ON AC
- If **temperature < 26°C**, turn OFF AC

7.4 Alexa Voice Commands for Relay Control

Integrating Alexa with Home Assistant

Alexa can be used to control relays via voice commands. To set this up:

Step 1: Install Home Assistant Cloud

- Go to **Home Assistant → Settings → Home Assistant Cloud**
- Sign in with a Nabu Casa account (required for Alexa integration)

Step 2: Add Home Assistant to Alexa

- Open the **Amazon Alexa App**
- Go to **Skills & Games** → Search for **Home Assistant**
- Click **Enable** and sign in with Home Assistant credentials

Step 3: Discover Devices

- In Alexa App, go to **Devices → Discover**
- Alexa will automatically find Home Assistant relays

Step 4: Control Devices Using Voice Commands

- Say **"Alexa, turn on the AC"** to activate the relay
- Say **"Alexa, turn off the AC"** to deactivate the relay

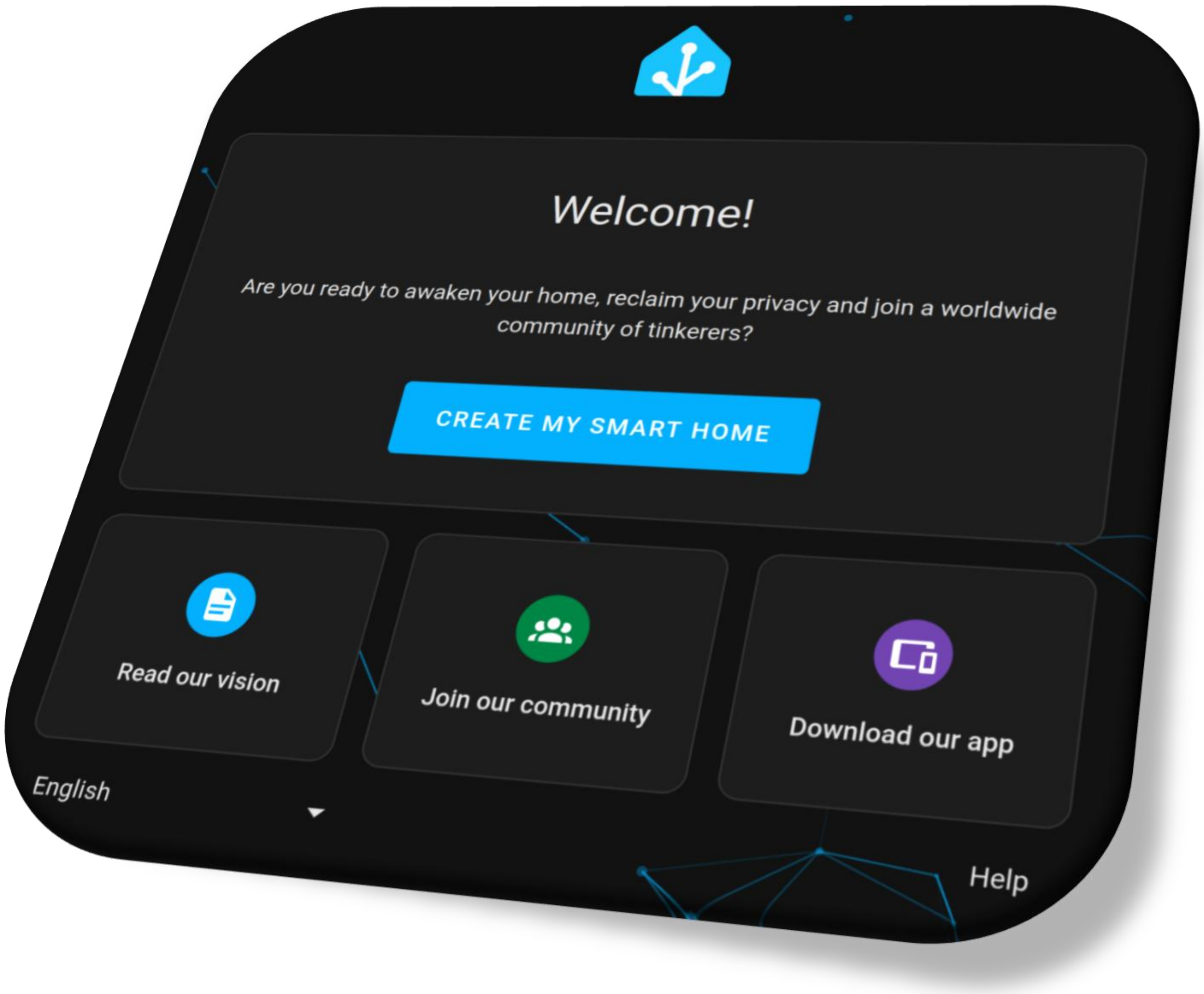
7.5 Security and Encryption in Connectivity

Since the system is connected to the internet, it is essential to ensure **secure communication** between ESP32, Home Assistant, and Alexa.

- **Security Measures in the Project:**

Wi-Fi Encryption: Use **WPA2 encryption** for securing the Wi-Fi connection. **MQTT with Authentication:** If using MQTT, enable **username/password authentication**. **Firewall & Port Restriction:** Prevent unauthorized access to Home Assistant. **SSL Encryption:** Use **HTTPS** to encrypt communication between Home Assistant and Alexa. **Strong Passwords:** Use strong passwords for Home Assistant and Alexa accounts.

Conclusion


Welcome!
Are you ready to awaken your home, reclaim your privacy and join a worldwide community of tinkerers?
CREATE MY SMART HOME
Read our vision
Join our community
Download our app
English
Help

8.1 Summary of Project Success

The **ESP32-based Home Automation System with Home Assistant and Alexa Integration** successfully met its objectives. The system allows:

- **Real-time Monitoring** – Temperature data from the **DHT11 sensor** is continuously monitored and displayed on Home Assistant.
- **Smart Appliance Control** – The **relay module** enables automatic and manual control of electrical appliances (e.g., AC, lights).
- **Temperature-Based AC Automation** – The system automatically turns ON/OFF AC based on room temperature.
- **Voice-Control with Alexa** – Users can control appliances using simple voice commands via **Amazon Alexa**.
- **User-Friendly Dashboard** – The **Home Assistant UI** provides an intuitive platform for managing home automation.
- **Secure Wireless Connectivity** – The ESP32 connects over **Wi-Fi** to communicate with Home Assistant.
- **Scalability** – Additional sensors, relays, and automation rules can be integrated into the system for future upgrades.

Overall, the project demonstrates a **cost-effective, efficient, and smart home automation solution**.

8.2 Limitations of the Current System

Despite its success, the system has some limitations that can be improved in future iterations:

- **Dependence on Wi-Fi** – The system **requires a stable internet connection** for remote access and Alexa integration. If Wi-Fi goes down, remote control will be unavailable.
- **Limited Local Voice Processing** – Alexa **relies on cloud services** for voice commands. A local, offline voice processing system (e.g., Mycroft AI) could improve privacy and reliability.

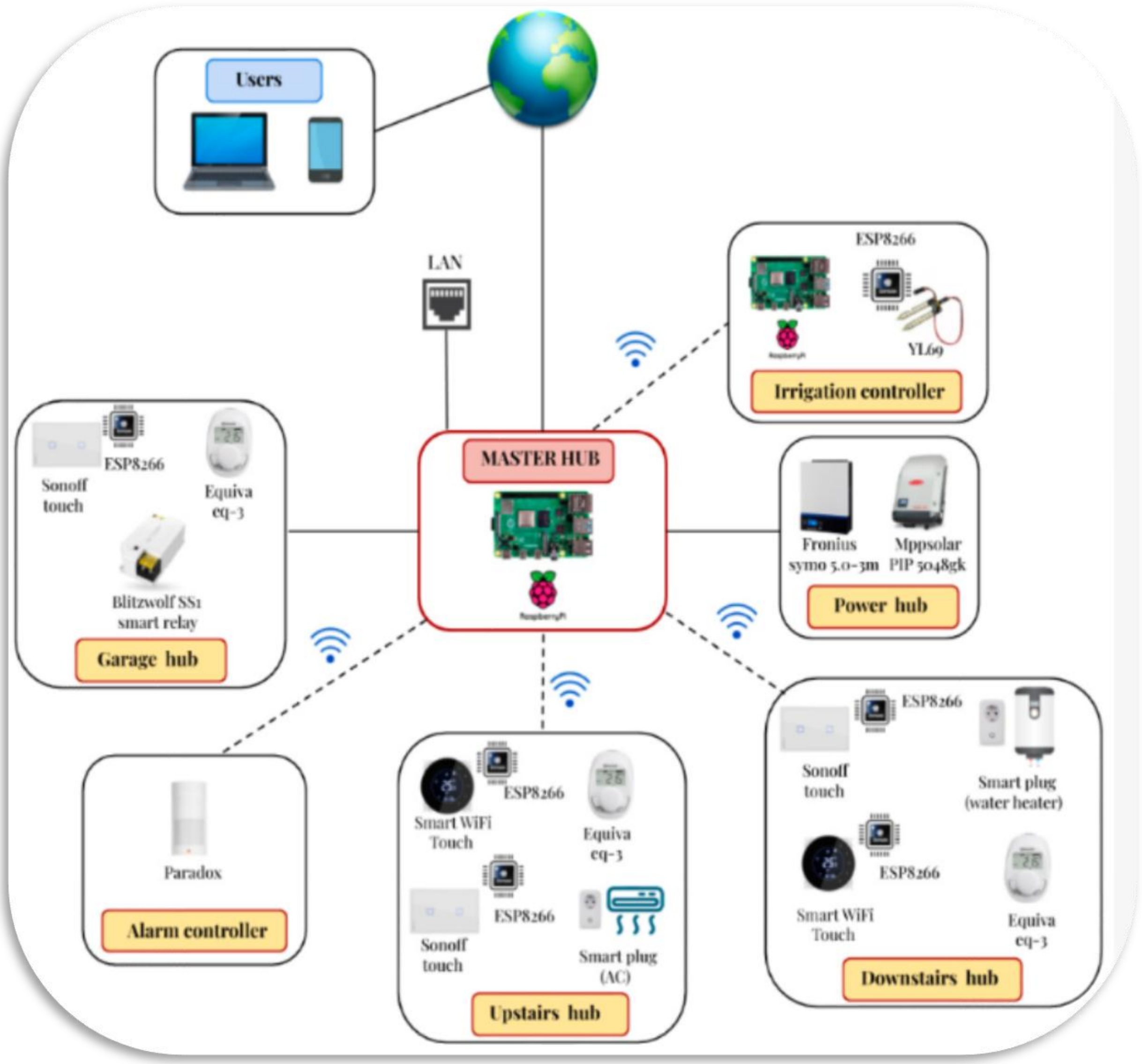
- **Basic Security Measures** – While the system has **password protection and SSL encryption**, it could be enhanced with **firewalls, VPNs, or two-factor authentication** for better security.
- **Fixed Temperature Thresholds** – The AC control is based on **static temperature values**. A more **adaptive algorithm (e.g., machine learning-based automation)** could optimize energy efficiency.
- **Single Sensor Integration** – Currently, the system only includes a **DHT11** sensor. Adding **motion sensors, gas sensors, or door sensors** could enhance automation capabilities.

8.3 Key Learnings from the Project

This project provided valuable insights into **IoT, embedded systems, and home automation**. The key learnings include:

- **ESPHome for IoT Device Configuration** – Understanding how **ESPHome simplifies ESP32 programming** using YAML configurations.
- **Home Assistant Setup & Automation** – Learning how to **integrate and automate devices** within Home Assistant.
- **Wi-Fi & MQTT Communication** – Understanding how **ESP32 communicates with Home Assistant using Wi-Fi and MQTT protocols**.
- **Relay & Sensor Interfacing** – Gaining hands-on experience with **DHT11 sensor and relay module control**.
- **Voice Integration with Alexa** – Learning how to **connect Home Assistant with Amazon Alexa** for voice-controlled automation.
- **Security Considerations in IoT** – Exploring **best practices for securing smart home systems** against cyber threats.
- **Debugging & Troubleshooting** – Learning how to **analyze logs, debug ESPHome firmware, and fix connectivity issues**.

Future Enhancements



9. Future Enhancements

As technology evolves, the home automation system can be enhanced with **advanced sensors, AI-driven automation, and improved hardware** for better efficiency, security, and intelligence.

9.1 Adding More Sensors

To improve safety, security, and environmental monitoring, additional sensors can be integrated into different rooms of the house:

❖ **Kitchen:**

- **Gas Leakage Sensor (MQ-5 or MQ-9):** Detects any gas leaks and triggers an alert to Home Assistant and Alexa.
- **Gas Measurement Sensor:** Monitors the level of gas usage and consumption to prevent wastage.
- **Smoke Sensor (MQ-2 or MQ-135):** Detects smoke and potential fire hazards, triggering alarms and ventilation systems.

❖ **Living Room:**

- **Temperature and Humidity Sensor (DHT22 or BME280):** Provides more accurate readings for climate control.
- **Air Quality Sensor (MQ-135):** Monitors indoor air quality and detects harmful gases like CO2 and VOCs.

❖ **Bedrooms:**

- **Motion Sensors (PIR Sensor):** Detects movement for security automation and energy saving.
- **Smart Lighting Control (LDR Sensor):** Adjusts lighting based on ambient brightness.

❖ **Entire House:**

- **Door/Window Sensors:** Sends alerts when doors or windows are opened unexpectedly.
- **Water Leakage Sensor:** Detects water leaks in areas like bathrooms or kitchens to prevent flooding.

9.2 Implementing Power Consumption Monitoring

- **Smart Energy Monitoring Modules (PZEM-004T, HLW8012):**
  - Track real-time power consumption for each appliance.
  - Display energy usage on the Home Assistant dashboard.
  - Send alerts when an appliance consumes excessive energy.
- **Integration with Solar Panels & Battery Storage:**
  - Monitor energy generation from solar panels.
  - Optimize battery usage to reduce electricity bills.

9.3 Smart Scheduling for Home Appliances

- **AI-Driven Automation:**
  - Use **machine learning models** on Raspberry Pi 5 to analyze user habits.
  - Automatically schedule appliance usage based on past patterns.
- **Time-Based Automation:**
  - Set schedules for lights, fans, and AC based on daily routines.
  - Example: AC turns on **30 minutes before bedtime** for comfort.
- **Occupancy-Based Control:**
  - Motion sensors can turn off lights and appliances when no one is in the room.
  - Example: If no motion is detected for **10 minutes**, turn off lights.

9.4 AI-Based Automation Using Raspberry Pi 5

System Architecture with Raspberry Pi 5 (12GB RAM):

- **Raspberry Pi 5 as the Central Home Controller:**
  - Runs **Home Assistant and Machine Learning models** for advanced automation.
  - Handles **multiple rooms and devices** with higher processing power.

    - Stores logs and historical data for predictive automation.

9.4.1 Room-Specific Automation:

- ❖ **Kitchen:**

  - Raspberry Pi **monitors gas sensors** and shuts off the gas valve if a leak is detected.
  - AI models predict **cooking time and gas usage patterns** for energy efficiency.

- ❖ **Living Room:**

  - **ESP32 controls all smart devices** (lights, fans, TV, air purifier).
  - **AI-based climate control:** Raspberry Pi **analyzes air quality and temperature** to optimize ventilation.

- ❖ **Bedrooms:**

  - **Smart sleep tracking** using Raspberry Pi with motion sensors and temperature adjustments.
  - **Wake-up automation:** AI gradually turns on lights and plays soft music in the morning.

- ❖ ***Entire Home Security:***

  - **Facial Recognition Cameras** detect authorized and unauthorized persons.
  - **AI analyzes real-time security footage** and sends alerts for unusual activities.
  - **All sensors are unlocked and activated** when the system is armed for security mode.

Final Thoughts

By integrating **Raspberry Pi 5 with ESP32**, the home automation system will become more **intelligent, efficient, and secure**. With **AI-powered automation, smart sensors, and power monitoring**, the system can **reduce energy costs, improve safety, and provide a seamless user experience**.

References

❖ **ESP32 MICROCONTROLLER**

- Espressif Systems. (2023). ESP32 Datasheet. URL: https://www.espressif.com/en/products/socs/esp32/resources
- Random Nerd Tutorials. (2023). ESP32 Pinout Reference. URL: https://randomnerdtutorials.com/esp32-pinout-reference-gpios/
- ESPHome. (2023). ESPHome Official Documentation. URL: https://esphome.io/

❖ **DHT11 Temperature and Humidity Sensor**

- D-Robotics. (2010). DHT11 Datasheet. URL: https://www.mouser.com/datasheet/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf
- Arduino. (2023). DHT11 Library and Tutorial. URL: https://forum.arduino.cc/t/dht11-tutorial/449743

❖ **Relay Modules**

- Songle Relay. (2023). SRD-05VDC-SL-C Relay Datasheet. URL: https://components101.com/relays/srd-05vdc-sl-5v-relay-pinout-datasheet
- Last Minute Engineers. (2023). Interfacing Relays with Microcontrollers. URL: https://lastminuteengineers.com/relay-module-interfacing/

❖ **Home Assistant Integration**

- Home Assistant. (2023). Official Documentation. URL: https://www.home-assistant.io/docs/
- VMware. (2023). VMware Workstation Player Setup Guide. URL: https://docs.vmware.com/en/VMware-Workstation-Player/

❖ **Alexa Integration**

- Amazon. (2023). Alexa Smart Home Skill API Documentation. URL: https://developer.amazon.com/en-US/docs/alexa/smarthome/understand-the-smart-home-skill-api.html
- Nabu Casa. (2023). Home Assistant Cloud for Alexa. URL: https://www.nabucasa.com/

❖ **Power Supply and Circuit Design**

- Texas Instruments. (2020). Buck Converter Design Basics. URL: https://www.ti.com/lit/an/slva477b/slva477b.pdf
- All About Circuits. (2023). Power Supply Design for IoT Devices. URL: https://www.allaboutcircuits.com/

❖ **ESPHome Configuration**

- ESPHome GitHub Repository. (2023). YAML Configuration Examples. URL: https://github.com/esphome/esphome
- Home Assistant Community. (2023). ESPHome Automation Templates. URL: https://community.home-assistant.io/

❖ **Security and Encryption**

- OWASP. (2023). IoT Security Guidelines. URL: https://owasp.org/www-project-internet-of-things/
- Let’s Encrypt. (2023). SSL/TLS Encryption for Home Assistant. URL: https://letsencrypt.org/docs/

❖ **Future Enhancements (AI, Raspberry Pi 5)**

- Raspberry Pi Foundation. (2023). Raspberry Pi 5 Technical Specifications. URL: https://www.raspberrypi.com/documentation/
- Hackster.io. (2023). Machine Learning on Edge Devices. URL: https://www.hackster.io/projects/tags/machine-learning

❖ **General IoT and Home Automation**

- IEEE Xplore. (2023). IoT System Design Research Papers. URL: https://ieeexplore.ieee.org/Xplore/home.jsp

❖ **Additional Resources**

GitHub Repositories

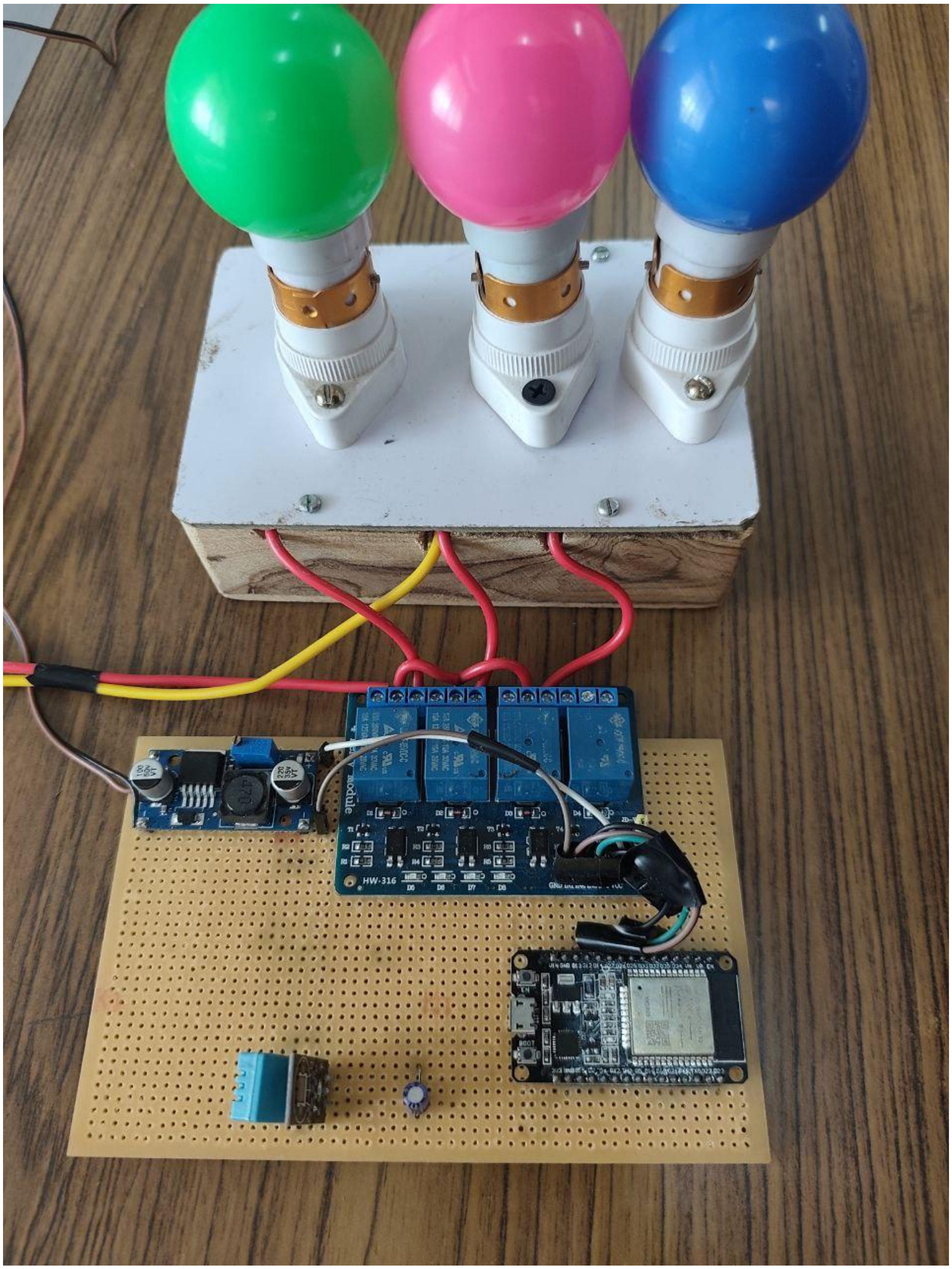
- Home Assistant Core: URL: https://github.com/home-assistant/core
- ESPHome Examples: URL: https://github.com/esphome/esphome-configs

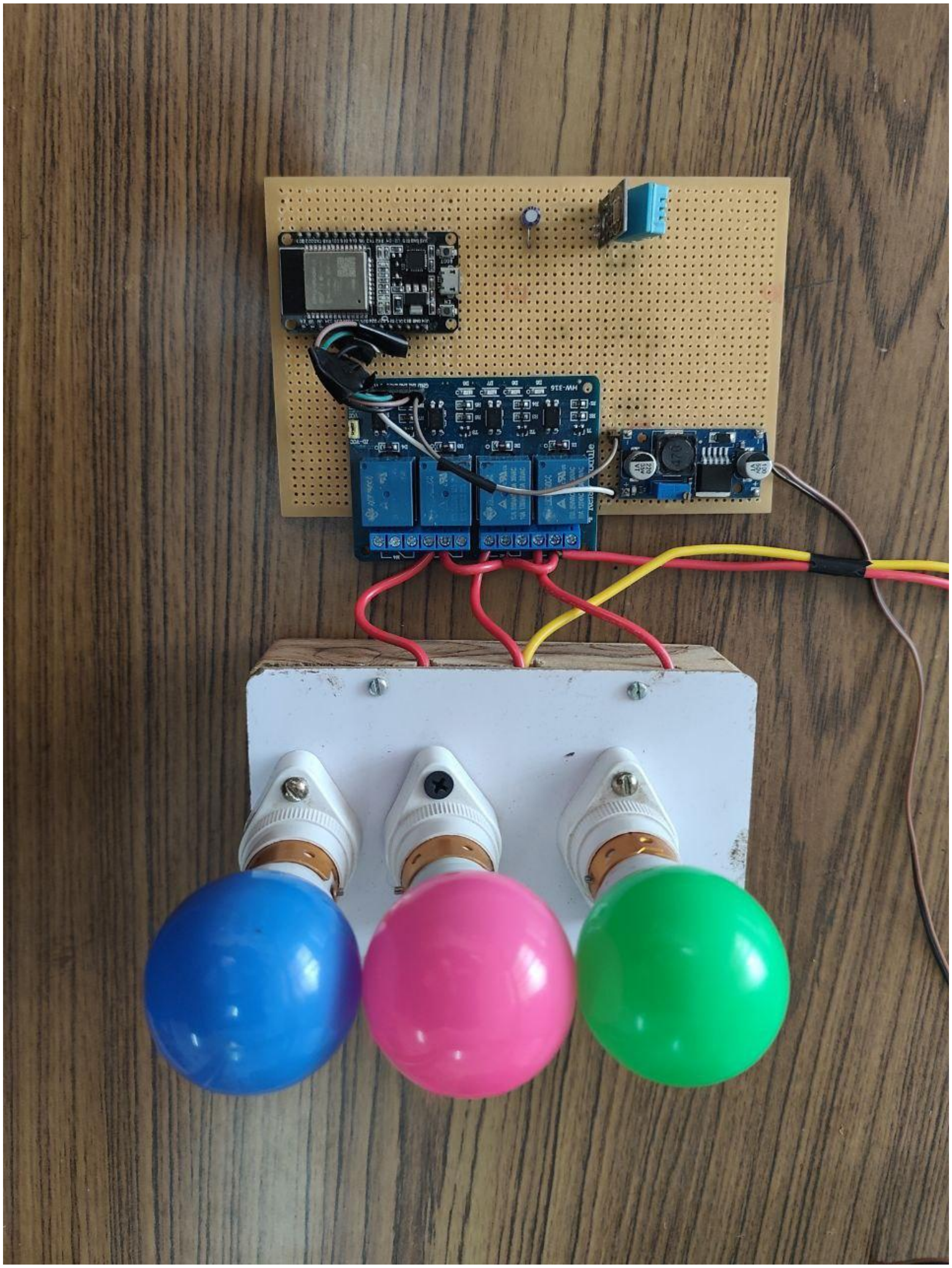
Communities

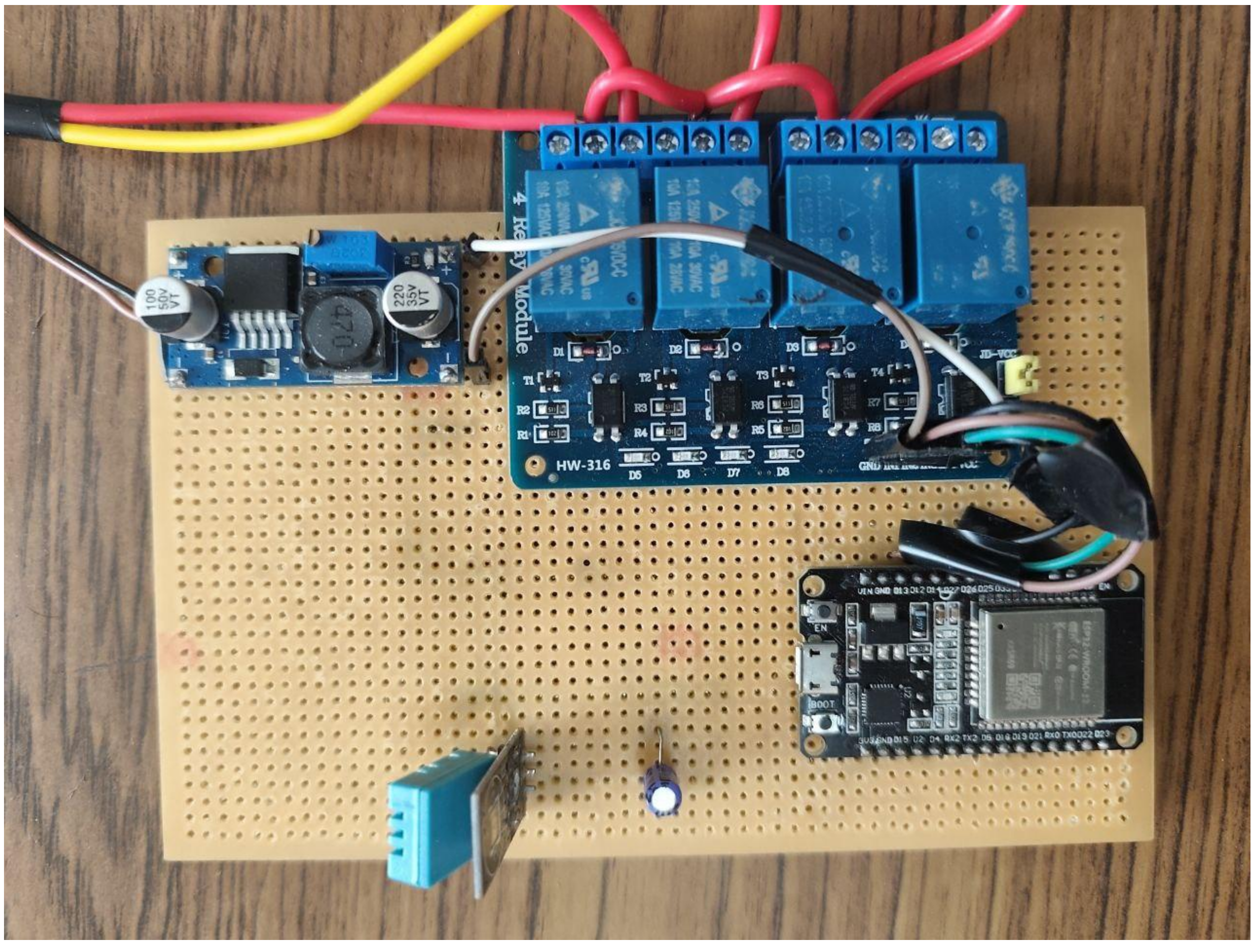
- Reddit: r/homeassistant URL: https://www.reddit.com/r/homeassistant/
- Stack Overflow: Tags for ESP32 URL: https://stackoverflow.com/questions/tagged/esp32

Photography

ESPHome


HW-316
module
BOOT




4 Relay Module
HW-316
EN
BOOT