Table of Contents

Introduction

⚠ This document is a work in progress. The content is subject to change.

Overview

This project aims to develop a **decentralized system** for **Controlled Environment Agriculture** (CEA).

The system has a modular structure, where each module has its own sub-network of accessory devices, such as sensors and actuators, that autonomously monitor and control environmental conditions without requiring a central hub.

The system prioritizes interoperability, enabling the integration of devices from different manufacturers and the replacement of existing ones without major updates. It also implements security measures to ensure safe data transmission and access control, maintaining system functionality and scalability.

Motivation

Advancements in IoT-based monitoring and control systems have improved automation in agriculture, increasing efficiency and productivity.

Precision Agriculture enhances field farming by using sensor data, GPS, and automated equipment to precisely manage inputs like water, fertilizers, and pesticides. *Controlled-Environment Agriculture* extends this approach by eliminating limitations such as external environmental factors, soil variability, and large-scale deployment challenges through enclosed systems where temperature, humidity, light, and CO_2 are actively regulated. This allows for year-round production, reduces reliance on weather conditions, and ensures consistent crop yields, making it more reliable than traditional open-field farming.

However, many farming automation startups struggle to succeed, primarily due to the high cost of research and development. Building advanced automation systems requires substantial investment, which raises food production costs and makes it difficult for startups to compete with traditional farms that operate at lower expenses.

A major factor driving these costs is the reliance on centralized servers or cloud resources for data processing and control. This setup requires complex communication infrastructure, increases operational expenses, and complicates scalability, particularly for larger farms. Additionally, centralized systems create a single point of failure - if the central server or cloud service fails or is compromised, the entire operation can be disrupted. For example, an AWS IoT outage in 2020 caused significant downtime for applications relying on AWS IoT Core, demonstrating the risks of centralized dependency.

Another issue is the difficulty of integrating components from different manufacturers. Automated farms use a variety of sensors, actuators, and equipment, but differences in APIs and communication protocols can make it challenging to connect these systems smoothly.

Discussion

The system leverages the Matter protocol, which ensures compatibility between devices from different manufacturers. This standardisation simplifies integration, replacement, and future expansion of devices. However, the high costs associated with Matter certification – including a $7,000 annual membership fee for the Connectivity Standards Alliance (CSA) and $3,000 per product certification – pose significant challenges for small manufacturers and farms operating on tight budgets.

Future advancements in this system could include the adoption of microfluidics within hydroponic systems. Such developments would enhance resource efficiency and allow for precise control over environmental conditions tailored to specific crops. These innovations align with the system's modular design philosophy, which prioritises adaptability and scalability in agricultural automation.

References

[1] L. Adenauer, "Up, up and away! The Economics of Vertical Farming," *Journal of Agricultural Studies*, doi: 10.5296/jas.v2i1.4526 (https://doi.org/10.5296/jas.v2i1.4526).

[2] U.S. Department of Agriculture, Office of the Chief Scientist, and U.S. Department of Energy, Bioenergy Technologies Office, Workshop Report: Research and Development Potentials in Indoor Agriculture and Sustainable Urban Ecosystems, Washington, D.C.,

Feb. 2019. Online (https://www.usda.gov/sites/default/files/documents/indoor-agriculture-workshop-report.pdf).

[3] People, power costs keep indoor farming down to Earth, Finance & Commerce, Associated Press, May 14, 2018. Online (https://finance-commerce.com/2018/05/people-power-costs-keep-indoor-farming-down-to-earth/)

[4] G. Johnson, "AeroFarms files for Chapter 11 bankruptcy protection," *Produce Blue Book*, Jun. 09, 2023. Online (https://www.producebluebook.com/2023/06/08/aerofarms-files-for-chapter-11-bankruptcy-protection/).

[5] B. C. Baraniuk, "Lean times hit the vertical farming business," Jul. 17, 2023. Online (https://www.bbc.com/news/business-66173872).

[6] S. Bökle, D. S. Paraforos, D. Reiser, and H. W. Griepentrog, "Conceptual framework of a decentral digital farming system for resilient and safe data management," *Smart Agricultural Technology*, Volume 2, Feb. 2022. Online (https://www.sciencedirect.com/science/article/pii/S2772375522000065).

[7] U. Shafi, R. Mumtaz, J. García-Nieto, S. A. Hassan, S. A. R. Zaidi, and N. Iqbal, "Precision Agriculture Techniques and Practices: From Considerations to applications," *Sensors*, vol. 19, no. 17, p. 3796, Sep. 2019, doi: 10.3390/s19173796 (https://doi.org/10.3390/s19173796).

[8] D. K. S. Karanam Desai, "Automation in Agriculture: a Study," *International Journal of Engineering Science*, vol. 2(2), Jun. 2016. Online (https://www.researchgate.net/publication/304650250_Automation_in_Agriculture_A_Study).

[9] "Distributed or centralized mobility?," *IEEE Conference Publication | IEEE Xplore*, Nov. 01, 2009. Online (https://ieeexplore.ieee.org/abstract/document/5426302/).

[10] D. Belli, P. Barsocchi, and F. Palumbo, "Connectivity standards alliance matter: state of the art and opportunities," *Internet of Things*, vol. 25, doi: 10.1016/j.iot.2023.101005 (https://doi.org/10.1016/j.iot.2023.101005).

[11] G.-J. Ra and I.-Y. Lee, "A study on KSI-based authentication management and communication for secure smart home environments," *KSII Transactions on Internet and Information Systems*, vol. 12, no. 2, Feb. 2018, doi: 10.3837/tiis.2018.02.021 (https://doi.org/10.3837/tiis.2018.02.021).

[12] R. Speed, “AWS admits to ‘severely impaired’ services in US-EAST-1, can’t even post updates to Service Health Dashboard,” *The Register*, Nov. 26, 2020. Online (https://www.theregister.com/2020/11/25/aws_down/).

[13] V. S, V. E. R D, V. A C, V. A, and S. L. S, “A STUDY ON DEVELOPMENT OF CROPS BY FOGPONIC SYSTEM USING COCO COIR,” *International Research Journal of Engineering and Technology (IRJET)*, vol.7. Online (https://www.irjet.net/archives/V7/i4/IRJET-V7I41142.pdf).

[14] V. O. Oner, *Developing IoT Projects with ESP32 - Second Edition*, O’Reilly Online Learning. Online (https://learning.oreilly.com/library/view/developing-iot-projects/9781803237688/Text/Chapter_3.xhtml#_idParaDest-48).

[15] Alliance Marketing, “Peeking Under the Hood of Your Matter Smart Home - CSA-IOT,” *CSA-IOT*, Aug. 28, 2024. Online (https://csa-iot.org/newsroom/peeking-under-the-hood-of-your-matter-smart-home/).

[16] “Technical documentation.” Online (https://docs.nordicsemi.com/bundle/ncs-2.6.1/page/matter/chip_tool_guide.html).

[17] “Programming Guide - ESP32 - Espressif’s SDK for Matter latest documentation.” Online (https://docs.espressif.com/projects/esp-matter/en/latest/esp32/).

[18] “Commissioning | Overview Guides | Silicon Labs Matter | V2.2.1 | Silicon Labs.” Online (https://docs.silabs.com/matter/2.2.1/matter-overview-guides/matter-commissioning).

[19] “ESP32-H2 Thread/Zigbee & BLE 5 SOC | Espressif Systems.” Online (https://www.espressif.com/en/products/socs/esp32-h2).

[20] “Introduction to Matter | Introduction to Matter | Silicon Labs Matter | V2.1.1 | Silicon Labs.” Online (https://docs.silabs.com/matter/2.1.1/matter-fundamentals-introduction/).

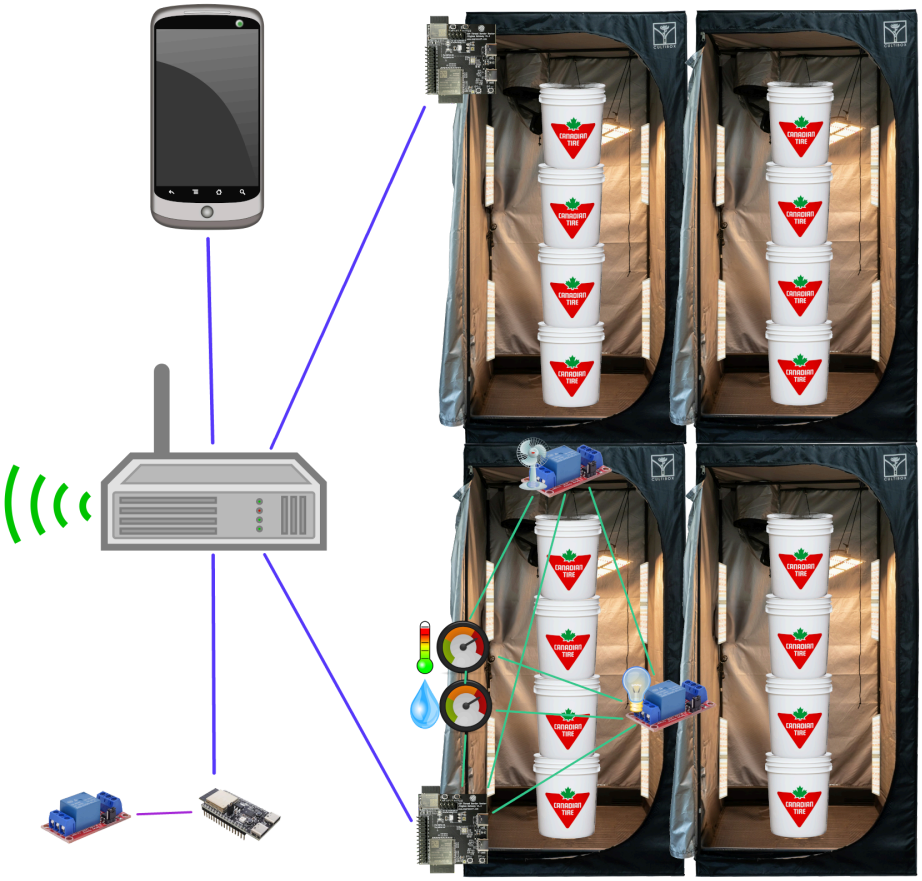
[21] “Matter FAQs | Frequently Asked Questions - CSA-IOT,” *CSA-IOT*. Online (https://csa-iot.org/all-solutions/matter/matter-faq/).

[22] “Thread benefits.” Online (https://www.threadgroup.org/What-is-Thread/Thread-Benefits).

[23] “OpenThread Border Router,” *OpenThread*. Online (https://openthread.io/guides/border-router).

System Design

Architecture Overview



This automated **Controlled Environment Agriculture** system is designed with a modular structure composed of independent **Grow Chambers** that include the following components:

- **Root Chamber** contains the plant's root system and nutrient solution.
- **Lighting** provides artificial light for photosynthesis.
- **Ventilation** regulates temperature, humidity, and CO_2 levels.

Each Grow Chamber is an independent unit designed to maintain a controlled environment for plant growth. Environmental conditions are regulated through a mesh

network of **Accessory Devices**:

- **Sensors** monitor temperature, humidity, light intensity, and CO_2 levels.
- **Actuators** adjust conditions by controlling fog generation, ventilation, and lighting.

The **Orchestrator** devices allow farmers to set up and manage Grow Chambers, while **Controller** devices provide an interface for monitoring and control.

Network Communication

Thread protocol enables low-power communication between the devices and ensures network reliability. If one device fails, the rest of the network remains operational.

Wi-Fi and Thread integration through **Thread Border Routers** allows remote monitoring and control through **Controller** devices.

Matter enables secure and standardized communication over Wi-Fi, Ethernet, and Thread, allowing sensors, actuators, and controllers to work together without compatibility issues.

Development Plan

Sprint 0: System Design

The primary goal of this sprint is to design the system architecture.

Timeline: September 6, 2024 - October 6, 2024

- ✅ Design system architecture
- ✅ Design network infrastructure
- ✅ Define hardware and software components

Sprint 1: Accessory Device Development

The primary goal of this sprint is to develop and validate an accessory device, a temperature sensor.

Timeline: October 6, 2024 - November 7, 2024

- ✅ Set up ESP-IDF and ESP-Matter Development Environment
- ✅ Assemble hardware for BMP280-based Sensor
- ✅ Develop firmware for BMP280 Sensor
- ✅ Test I2C Communication
- ✅ Configure Matter Controller on Raspberry Pi
- ✅ Test Matter Device Commissioning (BLE & Wi-Fi Pairing)

Sprint 2: Grow Chamber Assembly

The primary goal of this sprint is to assemble the Grow Chamber.

Timeline: November 8, 2024 - December 10, 2024

- ✅ Assemble grow tent

- ✅ Install exhaust fan
- ✅ Install lighting
- ✅ Assemble root chamber
- ✅ Integrate a temperature sensor
- ✅ Integrate a Mist Maker

Sprint 3: Orchestrator Device Development

The primary goal of this sprint is to develop and validate an orchestrator device.

Timeline: January 8, 2025 - February 5, 2025

User Stories:

- US1: Orchestrator Device Assembly & Configuration
- US2: Communication Reliability Across Protocols
- US3: Real-Time Sensor Monitoring
- US4: Actuator Control Based on Sensor Inputs
- US5: Orchestrator Local Data Storage
- US6: Device Discovery & Integration

Functional Requirements:

- FR1: Orchestrator Device Management
- FR2: Sensor Monitoring & Alerts
- FR3: Actuator Control & Automation
- FR4: Device Discovery & Integration

Non-Functional Requirements:

- NFR1: Performance

- NFR2: Reliability
- NFR4: Scalability

Sprint 4: Controller Development

The primary goal of this sprint is to develop and validate a controller device.

Timeline: February 6, 2025 - February 19, 2025

User Stories:

- US7: Controller Device Assembly & Configuration
- US8: Environment Profiles in Orchestrator
- US9: Selecting Environmental Profiles and Settings
- US10: Remote Monitoring via Controller Application
- US11: Real-Time Alerts for Environmental Deviations

Functional Requirements:

- FR5: Environmental Profiles & Grow Chamber Management
- FR6: Remote Monitoring & Control
- FR7: User Authentication & Access Control

Non-Functional Requirements:

- NFR3: Security & Access Control
- NFR6: Usability & User Experience

Sprint 5: System Integration

The primary goal of this sprint is to integrate all components into the system.

Timeline: February 20, 2025 - March 5, 2025

User Stories:

- US12: Integration of Third-Party Agricultural Sensors
- US13: Grow Chamber Management & Scaling
- US14: Development of Humidity Sensor
- US15: Development of CO_2 Sensor

Functional Requirements:

- FR5: Environmental Profiles & Grow Chamber Management
- FR6: Remote Monitoring & Control

Non-Functional Requirements:

- NFR5: Interoperability
- NFR7: Compliance & Standards

Requirements

Functional Requirements

FR1. Device Management

- The system must support the **ESP32-C6** as Controller hardware.
- The Controller must be able to **continuously monitor device connectivity** across Matter, Thread, and Wi-Fi networks.
- The system must provide **fallback communication modes** (e.g., switching from Thread to Wi-Fi) in case of network failure.

FR2. Sensor Monitoring & Alerts

- The system must **collect real-time sensor data** from temperature, humidity, and CO_2 sensors.
- The system must generate **alerts** when sensor values exceed predefined thresholds.
- Users must be able to view **historical trends of sensor data** for diagnostics.
- The system must allow **manual sensor calibration** to maintain accuracy.

FR3. Actuator Control & Automation

- The system must **automatically trigger actuators** (mist maker, exhaust fan, LED grow lights) based on sensor inputs.
- Growers must be able to **manually override actuator actions** via the Controller.
- The system must support **custom automation rules** based on environmental conditions.

FR4. Device Discovery & Integration

- The Controller must **detect and integrate new sensors and actuators** upon activation.

- The system must support **Matter, Thread, and Wi-Fi** for seamless device integration.
- Users must be able to **manually configure devices** if they are not automatically recognized.

FR5. Environmental Profiles & Grow Chamber Management

- The system must support **predefined and customizable environmental profiles** for different plant types.
- Growers must be able to **apply environmental settings** to one or multiple Grow Chambers.
- The system must allow users to **add, remove, and configure multiple Grow Chambers**.

FR6. Remote Monitoring & Control

- The system must provide a **web-based and mobile Controller application** for remote monitoring.
- Users must be able to view **real-time sensor and actuator data**.
- The system must send **real-time alerts via email, SMS, or push notifications**.

FR7. User Authentication & Access Control

- The system must implement **role-based access control (RBAC)** (Administrator, Technician, Grower).
- Remote access to the system must require **authentication**.
- All **user actions (settings changes, manual overrides) must be logged**.

Non-Functional Requirements

NFR1. Performance

- The system must process sensor data and actuator responses **within 1 second**

- Controller must be capable of handling **simultaneous data streams** from multiple sensors and actuators without performance degradation

NFR2. Reliability

- The system must be **capable of offline operation** in case of network disruptions
- All sensor readings and logs must be stored **locally for at least 30 days**
- Automatic device reconnection implemented when network connectivity is restored

NFR3. Security & Access Control

- All data communications must be **encrypted**
- Remote access to the Controller must require authentication

NFR4. Scalability

- The system must be **modular**, allowing the addition new devices without major modifications
- Support for at least 20 Grow Chambers with independent environmental settings

NFR5. Interoperability

- The architecture must be designed to support **third-party sensors and actuators**
- The Controller must be able to **automatically detect and configure new devices**

NFR6. Usability & User Experience

- Controller must provide an **intuitive interface** that can be used with minimal training

NFR7. Compliance & Standards

- The system must support the **Matter 1.3 and Thread 1.2** IoT standards

Backlog

US1: Border Router

As a developer, I want to set up a Border Router so that Thread devices can communicate with external IP networks.

Acceptance Criteria:

- The Border Router must connect the Thread Network to external IP networks (Wi-Fi or Ethernet).
- The Border Router must run on ESP Thread Border Router board.
- The system must provide a Web GUI for configuring and monitoring Thread networks.
- The system must allow devices to communicate with the internet.

US2: Communication Reliability Across Protocols

As a Border Router, I want to ensure smooth communication between Matter, Thread, and Wi-Fi devices so that all system components remain connected.

Acceptance Criteria:

- The Border Router must **continuously monitor device connectivity** across Matter, Thread, and Wi-Fi networks.
- The system must **automatically reconnect devices** if they temporarily lose connection.
- The system must trigger an **alert** if a device remains disconnected beyond a defined threshold.
- The Border Router must provide **fallback communication modes** (e.g., switching from Thread to Wi-Fi) if a primary network fails.

US3: Real-Time Sensor Monitoring

As a Controller, I want to monitor all connected sensors in real-time so that I can adjust environmental conditions dynamically.

Acceptance Criteria:

- The Controller must collect **real-time sensor data** (humidity, temperature, CO_2, etc.).
- The system must alert when a sensor value goes outside the defined Environmental Profile range.
- The technician must be able to view **historical sensor data trends** for diagnosing system performance.
- The system must **automatically trigger corrective actions** based on sensor readings (e.g., activate mist maker if humidity drops).

US4: Actuator Control Based on Sensor Inputs

As a Controller, I want to trigger actuators (mist maker, exhaust fan, LED grow lights) based on sensor inputs so that plants receive optimal care.

Acceptance Criteria:

- The system must trigger the **mist maker** if humidity drops below the target range.
- The system must activate the **exhaust fan** if CO_2 levels exceed the defined threshold.
- The technician must be able to **manually override actuator actions** if needed.

US5: Controller Local Data Storage

As a Controller, I want to store sensor data locally so that I can continue operations even during network failures.

Acceptance Criteria:

- The Controller must **store all sensor readings and actuator logs locally** in case of network disruptions.
- The technician must be able to access locally stored data through a **diagnostic interface**.

US6: Controller Device

As a technician, I want to assemble and configure the Controller device so that growers can remotely monitor and control the system.

Acceptance Criteria:

- The Controller device must be assembled and flashed with the appropriate firmware.
- The mobile/web application must be deployed to connect with the Orchestrator.
- The controller must display real-time sensor data and allow actuator control.

US7: Device Discovery & Integration

As n Controller, I want to detect new accessory devices (sensors, actuators) and add them to the network so that they can be controlled efficiently.

Acceptance Criteria:

- The Controller must automatically **detect new sensors and actuators** when they are powered on and within the network range.
- The system must support **Matter, Thread, and Wi-Fi** communication protocols for seamless integration.
- The Controller must be able to **assign each new device** to a specific Grow Chamber.
- The technician can **manually configure detected devices** if they are not automatically recognized.
- The system must ensure that newly added devices **sync with predefined Environmental Profiles** for optimal operation.
- The system must notify the technician if a device **fails to connect or has a communication issue**.

US8: Environment Profiles in Controller

As a Controller, I want to define preset environmental configurations for different plant types so that growers can easily set up new crops.

Acceptance Criteria:

- The system must include **predefined Environmental Profiles** for various plant types (e.g., leafy greens, fruiting plants, herbs).
- The technician must be able to **create and customize new Environmental Profiles** based on specific crop needs.
- Each Environmental Profile must define:
  - **Target Humidity Range (%)**
  - **Target Temperature Range (°C/°F)**
  - **Target CO_2 Levels (ppm)**
  - **Light Intensity & Schedule**
  - **Ventilation Requirements**

US9: Selecting Environmental Profiles and Settings

As a grower, I want to select environmental settings for different crops so that I can optimize growth conditions.

Acceptance Criteria:

- The system allows the grower to **select from predefined Environmental Profiles** tailored for specific crops.
- Growers can **customize temperature, humidity, CO_2 levels, and light schedules** for each crop.
- The system must apply the **selected environmental settings to one or multiple Grow Chambers**.

US10: Remote Monitoring via Controller Application

As a grower, I want to remotely monitor grow chambers via a web application so that I can check the status of my crops from anywhere.

Acceptance Criteria:

- The web application must display **real-time data** for all Grow Chambers.
- Growers must be able to **view temperature, humidity, CO_2, and actuator status**.
- The interface must allow **quick switching between different Grow Chambers**.

US11: Real-Time Alerts for Environmental Deviations

As a grower, I want to receive real-time alerts when temperature, humidity, or CO_2 levels exceed acceptable thresholds so that I can take corrective actions.

Acceptance Criteria:

- The system must trigger **real-time alerts** when sensor values exceed or fall below predefined thresholds.
- The alert system must differentiate between **critical, warning, and informational notifications**.
- The system must log all alerts and corrective actions taken for future reference.

US12: Integration of Third-Party Agricultural Sensors

As a grower, I want to integrate third-party accessory devices into the system so that I can easily replace or add new devices.

Acceptance Criteria:

- The system must support **Matter, Thread, and Wi-Fi-compatible sensors** from third-party manufacturers.
- The grower must be able to **manually configure third-party devices**.
- The system must display **data from third-party sensors alongside built-in system sensors**.

- Alerts and automation rules must be **customizable for third-party sensors**.

US13: Grow Chamber Management & Scaling

As a grower, I want to manage Grow Chambers, so I can scale my output.

Acceptance Criteria:

- The system must allow growers to **add, remove, and configure multiple Grow Chambers**.
- The dashboard must provide a **centralized view of all active Grow Chambers**.
- The grower must be able to **duplicate existing chamber configurations** to streamline setup for new chambers.

US14: Development of Humidity Sensor

As a developer, I want to implement a humidity sensor so that the system can accurately measure and regulate humidity levels.

Acceptance Criteria:

- The system must successfully interface with the **DHT22 sensor** to collect **real-time humidity and temperature data**.
- The humidity sensor must provide readings within **±2% accuracy** in the target range.

US15: Development of CO_2 Sensor

As a developer, I want to implement a CO_2 sensor so that the system can monitor and regulate carbon dioxide levels in grow chambers.

Acceptance Criteria:

- The system must successfully interface with the CO_2 sensor to collect **real-time CO_2 data**.

- The CO_2 sensor must provide readings within **±50 ppm accuracy** within the operational range.

Grow Chamber

Overview

A **Grow Chamber** is a self-contained module within the system, designed to provide an independent, controlled environment for plant growth.

System Design

Commercially available grow tents for indoor plant cultivation function as individual **Grow Chambers**, providing structure and isolating the growing environment.

A bucket at the centre of the tent acts as a **Root Chamber**. It contains multiple **Plant Slots** that hold net pots filled with growing medium, where seeds are planted.

As the seeds germinate, their stems grow upward to form a canopy, while roots develop below. The **Fogponics System** delivers a fine, nutrient-rich mist to the roots, generated by an ultrasonic mist maker submerged in a water-nutrient solution.

Lighting panels positioned in the corners provide the necessary wavelengths for growth. The walls of the Tent and the Root Chamber are lined with Mylar, a reflective material that enhances light distribution for the plants.

The **Ventilation system** maintains airflow and regulates the temperature within the Grow chamber.

Hardware Components

The hardware components were sourced from easily accessible suppliers, such as Canadian Tire and Amazon.

Grow Chamber Assembly

Required Components:

- Grow Tent for indoor plant cultivation (24" x 24" x 48", 61cm x 61cm x 122cm)
- Full Spectrum Grow Light for Indoor Plants
- Inline Fan: 4-inch diameter
- Carbon Filter: 4-inch flange diameter, 14 inches long
- Ducting: 4-inch diameter, 8 feet long
- Rubber Coupler Connector: Compatible with a 4-inch diameter

Root Chamber Assembly

Required Components:

- Canadian Tire Plastic Food Grade Safe Bucket, 5-Gal/19-L (37.20 cm x 31.00 cm) or 7.5-L
- Mylar Films
- Ultrasonic Mist Maker, 400ml/h
- Brushless DC Fan, 12V, IP67, 120mm x 25mm
- Peristaltic Pump, 12V 3x5mm
- Silicone tube, 3x5mm 5m
- Aquarium air pump
- FS400-SHT3X Soil Temperature and Humidity Sensor (SHT35)
- Ultrasonic Distance Sensor, water-resistant
- 2" Net pots

Required Tools:

- Heat Gun
- Hole Saw, 1.5" diameter
- Wire Stripper
- 12V 1A AC/DC Power Adapter
- 12V DC Power Jack Connector Adapter, Male Plug & Female Socket Set, 5.5 X 2.1mm

Seed Planting

Required Components:

- Fertilizer
- Clay Pellets
- Organic cotton balls
- Arugula Seeds

Ventilation

The tent's internal volume is approximately 64,512 cubic inches, which converts to about 37.34 cubic feet: 32" × 32" × 63" ÷ 1,728 ≈ 37.34 ft^3

The 130 CFM inline duct fan can replace 37 cubic feet of air in about 17 seconds: 37 ft^3 ÷ 130 ft^3/min ≈ 0.28 minutes

For effective growing conditions, it's recommended to perform a full air exchange every one to three minutes. This ensures fresh air circulation, stable temperature, and proper humidity levels.

Automation

Overview

The system regulates the environment and resource availability based on plant growth stages, ensuring optimal conditions from germination to maturity. This involves developing components, assembling hardware, and integrating software.

Hardware

The system uses **Espressif** development boards: **ESP32-C6** and **ESP32-H2** for sensor data processing and actuator control, and the **Thread Border Router board** for communication with Thread-enabled devices and external networks. Sensors monitor temperature, humidity, and CO_2 levels, while actuators such as mist makers, exhaust fans, and LED grow lights adjust conditions based on automation rules.

Software

The system uses Thread (Thread) for low-power, mesh-based wireless communication between sensors, actuators, and control units. Matter (Matter) is used as the application layer protocol to enable device interoperability and secure communication.

Development is based on **ESP-IDF v5.3.2**, the official framework for ESP32, with **ESP-Matter v1.4** integrated for Matter device support. The environment is set up on **Ubuntu 24.04**, selected for its stability and compatibility with development tools.

Background

This section provides an overview of the relevant background information necessary to understand the project's context, including key concepts and technologies.

Thread

Overview

Thread is an IPv6-based networking protocol designed for low-power Internet of Things devices in an IEEE 802.15.4-2006 wireless mesh network.

Matter (Matter) devices can communicate with each other using Thread as a transport protocol.

OpenThread (OT) is an open-source implementation of the **Thread** networking protocol.

Resilience

The Thread Network prevents **single points of failure** through device redundancy and autonomous role transitions, but in topologies like a Thread Network Partition with only one Border Router, failure of that Border Router can disrupt external network access due to the lack of a backup. Devices in a Thread Network can communicate directly without an active Border Router. Having multiple Border Routers allows connections to one or more external networks, improving reliability and redundancy.

Architecture

OpenThread runs on various OS or bare-metal systems, including Linux and ESP32, due to its narrow abstraction layer and portable C/C++ design. It supports the following designs:

- **System-on-chip (SoC):** a single chip combines a processor and an 802.15.4 radio for Thread. In this design, both OpenThread and the device's main software (application layer) run on the same processor. This mode is used in **end devices** like sensors and actuators (ESP32-H2, ESP32-C6).
- **Radio Co-Processor (RCP):** the host processor runs the full OpenThread stack, while a separate Thread radio chip handles only the PHY and MAC layers of IEEE 802.15.4. They communicate using the Spinel protocol over SPI or UART. Since the host processor manages all networking tasks, it typically remains powered on, making this setup ideal for devices that prioritize performance over power efficiency. This mode is

used in **Thread Border Routers**, where a host processor, such as an ESP32-S3 or ESP32, is paired with an ESP32-H2 or ESP32-C6 as the Thread radio co-processor.

- **Network Co-Processor (NCP):** OpenThread runs on a Thread-enabled SoC, while the application software runs separately on a host processor. These two processors communicate using wpantund over SPI or UART, with the Spinel protocol managing the connection. Unlike RCP, this design allows the host processor to sleep, while the Thread-enabled chip stays active to maintain network connectivity. This design is beneficial for power-saving devices that require a constant network connection but do not need the host processor to remain active at all times.

Device Types

There are two types of Thread devices:

- **Minimal Thread Device (MTD)** communicates only with its FTD parent.
- **Full Thread Device (FTD)** communicates with other FTDs and its MTD children.

Device Roles

A Thread device can have multiple roles. For example, it can act as a Router, Border Router, and Commissioner simultaneously. See also Commissioning Roles ("Commissioning Roles" in "Thread Commissioning").

Router

A **Router** is an FTD ("Device Types" in "Thread") that provides routing services in a Thread Network. It forwards packets, maintains routing information, and serves as a Parent for End Devices ("End Device" in "Thread"). Additionally, Routers support Thread Commissioning (Thread Commissioning), handling device joining and security services.

A **Leader** is a special Router that manages a Thread Network Partition. Each Partition has one Leader, elected dynamically from the available Routers. If the Leader fails, another Router automatically takes over. The Leader is responsible for:

- Assigning and managing Router IDs.
- Maintaining Thread Network Data.

- Coordinating network operations.

End Device

An **End Device** is a device that connects to the Thread Network but does not forward packets like a Router ("Router" in "Thread"). Instead, it relies on a Parent Router for communication. End Devices are typically low-power devices, such as sensors or actuators, that do not need to maintain full network topology information.

Depending on their capabilities and power management strategies, End Devices are classified into the following types:

- **Router-Eligible End Device (REED)** is an FTD ("Device Types" in "Thread") that operates as an End Device but can be promoted to a Router ("Router" in "Thread") if the network requires additional routing capacity. It does not forward messages but maintains connections with Routers and supports Thread Commissioning (Thread Commissioning).
- A **Full End Device (FED)** is an FTD ("Device Types" in "Thread") that operates as an End Device and will never become a Router. Unlike a REED, a FED cannot be promoted to a Router.
- A **Minimal End Device (MED)** is an MTD ("Device Types" in "Thread") that keeps its radio on at all times and can communicate with its Parent Router whenever needed. MEDs do not forward messages or participate in routing.
- A **Sleepy End Device (SED)** is an MTD ("Device Types" in "Thread") that turns off its radio when idle to save power. It periodically wakes up to communicate with its Parent Router but does not forward messages.

Border Router

A **Thread Border Router** is a device that connects a Thread Network to external IP networks, such as Wi-Fi or Ethernet. It allows Thread devices to communicate with the internet or other smart home ecosystems.

OpenThread's implementation of a Border Router is called **OpenThread Border Router (OTBR)**.

Espressif (Espressif) provides the ESP Thread Border Router SDK ("ESP Thread Border Router Solution" in "Espressif") and hardware platforms ("ESP Thread Border Router

Solution" in "Espressif") for building Thread Border Routers.

OpenThread CLI

The **OpenThread CLI** is a command-line interface that provides configuration and management APIs for OpenThread.

It is used on OpenThread Border Routers (OTBR), Commissioners, and other Thread devices. The available commands depend on the device type. The `help` command prints all the supported commands.

References

- OpenThread Platforms (https://openthread.io/platforms)
- OpenThread Node Roles and Types (https://openthread.io/guides/thread-primer/node-roles-and-types)
- OpenThread Border Router (https://openthread.io/guides/border-router)
- Espressif OpenThread API (https://docs.espressif.com/projects/esp-idf/en/stable/esp32s2/api-guides/openthread.html)
- OpenThread CLI Overview (https://openthread.io/reference/cli)
- Espressif OpenThread CLI (https://github.com/espressif/esp-idf/tree/v5.4/examples/openthread/ot_cli)
- ESP Thread Border Router SDK (https://docs.espressif.com/projects/esp-thread-br/en/latest/)

Forming a Network

Active Operational Dataset

The **Active Operational Dataset** contains the configuration settings that Thread devices use to connect and operate within a specific Thread network:

- **Active Timestamp** – Determines dataset priority.
- **Channel** – PHY-layer channel for network communication.
- **Channel Mask** – Defines channels for network discovery and scanning.
- **Extended PAN ID** – Unique identifier for the Thread network.
- **Mesh-Local Prefix** – IPv6 prefix for local device communication.
- **Network Key** is a 128-bit key used to secure communication within the Thread network.
- **Network Name** – Human-readable network identifier.
- **PAN ID** – MAC-layer identifier for data transmissions.
- **PSKc** – Security key for network authentication.
- **Security Policy** – Specifies allowed and restricted security operations.

Thread Network Data

Thread Network Data is a collection of network-related information managed and distributed by the Leader ("Router" in "Thread") in a Thread network.

It includes the following details:

- Border Routers
- On-mesh prefixes
- External routes

- 6LoWPAN contexts
- Network commissioning parameters

The Leader ("Router" in "Thread") collects and updates Thread Network Data, distributing changes using MLE (Mesh Link Establishment) messages. Routers and REEDs store the full data, while End Devices (MTDs) ("End Device" in "Thread") can store either the full set or only a stable subset to save resources.

Thread Commissioning

The **Joiner ID** is derived from the Joiner Discerner if one is set; otherwise, it is derived from the device's factory-assigned EUI-64 using SHA-256. This ID also serves as the device's IEEE 802.15.4 Extended Address during commissioning. When the device joins a Thread network, it automatically receives the Network Key.

Commissioning Roles

The Thread Specification defines several commissioning roles within the **Mesh Commissioning Protocol (MeshCoP):**

- **Commissioner** manages device authentication and onboarding in a Thread network. It provides network credentials to new devices (**Joiners**) and can update network parameters or perform diagnostics. Types of Commissioners:
  - **On-Mesh Commissioner** operates inside the Thread network and has full control over commissioning.
  - **External Commissioner** resides outside the Thread network and connects via a **Border Agent** (e.g., a mobile app or cloud service).
  - **Native Commissioner** uses the same Thread interface as the mesh to manage commissioning.
- **Joiner** is a Thread device attempting to join a commissioned Thread network. It does not have network credentials and must authenticate with a Commissioner to gain access.
- **Border Agent (BA)** relays messages between an External or Native Commissioner and the Thread Network, ensuring secure communication between external networks and the Thread network.
- **Joiner Router** is a Router or REED that assists a Joiner in communicating with a Commissioner when the Joiner is not within direct range. It does not perform full routing but forwards Joiner messages to facilitate secure commissioning.

References

- Thread Commissioning (https://www.threadgroup.org/Portals/0/documents/support/CommissioningWhitePaper_658_2.pdf)
- OpenThread commissioning (https://docs.nordicsemi.com/bundle/ncs-latest/page/nrf/protocols/thread/overview/commissioning.html)

Matter

Overview

Matter is an **IP-based** IoT connectivity standard that defines a common application layer for secure and interoperable communication over **Wi-Fi**, **Thread** (Thread), and **Ethernet** networks.

Fabrics

A **Matter Fabric** is a private virtual network that connects Matter Devices and extends across Wi-Fi, Thread, and Ethernet physical networks. During the Matter commissioning process, controllers assign Fabric credentials to ensure secure integration.

Matter Commissioning

Matter Commissioning is the process of adding a **Commissionee** device to a **Fabric** by a **Commissioner** device.

Commissioning Process

Matter Commissioning involves the following steps:

1. The Commissioner retrieves the Onboarding Payload ("Onboarding Payload" in "Matter Commissioning") and discovers ("Device Discovery" in "Matter Commissioning") the Commissionee device.
2. Passcode-Authenticated Session Establishment (PASE) is used to establish encryption keys, securing communication between the Commissioner and Commissionee. This process also sets up an attestation challenge for verifying the device's authenticity.
3. The Commissioner verifies the Commissionee's authenticity through Device Attestation, ensuring it is a certified Matter device.
4. The Commissioner configures the Commissionee by providing essential information including network configuration settings, such as Wi-Fi credentials (SSID and passphrase) or Thread network credentials.
5. The Commissionee joins the operational network, unless it is already connected. The Commissioner or Administrator identifies or discovers the device's IPv6 address to enable further communication.
6. Certificate-Authenticated Session Establishment (CASE) is used to derive long-term encryption keys, securing all future unicast communication between the Commissioner or Administrator and the Commissionee.
7. The Commissioning process completes with an encrypted message exchange, confirming successful onboarding using CASE-derived encryption keys on the operational network.

Onboarding Payload

The **Commissionee** shares the **Onboarding Payload** with the **Commissioner** through a *QR Code*, *Manual Pairing Code*, or *NFC Tag*. It is composed of required and optional information. The following information may be included:

- **Version** specifies the payload format version, allowing future updates. It is 3 bits (0b000) for machine-readable formats and 1 bit (0b0) for Manual Pairing Codes.
- **Vendor ID** is a 16-bit identifier assigned by the Connectivity Standards Alliance (CSA) to uniquely identify a device manufacturer. It ensures that devices from different manufacturers can be distinguished within the Matter ecosystem.
- **Product ID** is a 16-bit identifier assigned by the manufacturer to differentiate between their products. It helps identify specific device models and is used in commissioning and attestation processes.
- **Custom Flow** is a 2-bit enumeration that specifies whether additional steps are required before commissioning. It informs the Commissioner if the device is ready for commissioning immediately, requires user interaction (such as pressing a button), or needs an external service interaction before being available for setup.
- **Discovery Capabilities Bitmask** is an 8-bit bitmask included in machine-readable formats. It indicates the discovery technologies (e.g., BLE, Wi-Fi, Thread) supported by the device, helping the Commissioner determine how to find and connect to it.
- **Discriminator** is a 12-bit identifier used to distinguish between multiple commissionable device advertisements. When a device enters commissioning mode, it broadcasts this value over BLE, Wi-Fi, or Thread, and it must match the value the device advertises. Each device should have a unique Discriminator to improve discovery and setup reliability. In machine-readable formats, the full 12-bit Discriminator is used, while in Manual Pairing Codes, only the upper 4 bits are included.
- **Passcode** is a 27-bit numeric value used to authenticate the device during commissioning, serving as proof of possession. It is encoded as an 8-digit decimal number ranging from 00000001 to 99999998, excluding invalid values. The Passcode is also used as a shared secret to establish a secure channel between the Commissioner and the device for further onboarding steps.

- **TLV Data** is optional, variable-length information stored in machine-readable formats using the Tag-Length-Value encoding. This data provides additional commissioning details.

Device Discovery

Device Discovery is the process where a **Commissioner** identifies a **Commissionee** before onboarding it to a **Fabric.** Devices announce their presence through the following methods:

- **BLE** is used by devices without network credentials to advertise their presence. The Commissioner scans for advertisements containing the Discriminator, Vendor ID, and Product ID to identify the correct device.
- **Wi-Fi / Ethernet** is used by devices already connected to a network, announcing themselves via mDNS.
- **Thread** is used by Thread-enabled devices, which register with a Thread Border Router using Service Registration Protocol (SRP).
- **User-Directed Commissioning (UDC)** allows a device to actively search for Commissioners, letting the user select one for onboarding.

Matter Controllers

Overview

A Matter Controller is a Node in a Matter Fabric that has the permissions needed to send commands or otherwise manage or coordinate other Nodes on the same Fabric.

Subscribing to events or attributes

Subscribing to an event or attribute means that its current state is automatically refreshed whenever changes occur within the Matter network. The events or attributes available for subscription are defined by the cluster in use.

Multiple subscriptions can run at once, and a single subscription can cover several events or several attributes - even if those come from different clusters. However, each subscription must be dedicated to either events or attributes; they cannot be mixed in one subscription.

Espressif

Overview

Espressif is a semiconductor company that develops SoCs, microcontrollers, and development boards for IoT applications.

Development Boards

Espressif's development boards are used in this system to enable wireless communication and automation:

- ESP32-H2-DevKitM-1 (https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32h2/esp32-h2-devkitm-1/index.html) is a development board with Thread and Bluetooth LE connectivity, designed for low-power applications.
- ESP32-C6-DevKitC-1 (https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32c6/esp32-c6-devkitc-1/index.html) is a development board with Wi-Fi 6, Bluetooth LE, and Thread connectivity.
- ESP32-DevKitM-1 (https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitm-1/index.html) is a development board with Wi-Fi and Bluetooth LE connectivity, designed for general-purpose applications.
- ESP Thread Border Router (https://docs.espressif.com/projects/esp-thread-br/en/latest/hardware_platforms.html) is a development board that acts as a Thread Border Router, enabling communication between Thread networks and Wi-Fi or Ethernet for external connectivity.

ESP-IDF Framework

ESP-IDF (**Espressif IoT Development Framework**) is Espressif's official IoT Development Framework for the ESP32, ESP32-S, ESP32-C and ESP32-H series of SoCs.

ESP Matter Solution

The **Espressif's SDK for Matter** is the official Matter (Matter) development framework for ESP32 series SoCs.

Wi-Fi-enabled ESP32, ESP32-C, and ESP32-S series can be used to build Matter Wi-Fi devices, with the ESP32-S series also supporting Ethernet connectivity via an external controller.

ESP32-H series SoCs, which support IEEE 802.15.4, are used for Matter-compatible Thread end devices.

ESP Thread Border Router Solution

The **Espressif Thread Border Router SDK** is a FreeRTOS-based solution built on the ESP-IDF ("ESP-IDF Framework" in "Espressif") and OpenThread (Thread) stack. It supports both Wi-Fi and Ethernet interfaces as the backbone link, combined with 802.15.4 SoCs for Thread communication.

The Wi-Fi-based Espressif Thread Border runs on two SoCs:

- The host Wi-Fi SoC, which runs OpenThread Border Router (ESP32, ESP32-S, or ESP32-C series SoC).
- The Radio Co-Processor (RCP), which enables the Border Router to access the 802.15.4 physical and MAC layers (ESP32-H series SoC).

Espressif also provides a single **ESP THREAD BR-ZIGBEE GW** board that integrates both the host SoC and the RCP into a single board.

Espressif also provides a **ESP Thread BR-Zigbee GW_SUB** daughter board for building an Ethernet-enabled Thread Border Router. It must be connected to a Wi-Fi-based ESP Thread Border Router.

References

- About Espressif (https://www.espressif.com/en/about)
- Espressif Development Boards (https://www.espressif.com/en/products/devkits/)

- ESP IoT Development Framework (https://www.espressif.com/en/products/sdks/esp-idf)

Development Environment Setup

ESP-IDF Setup

Overview

This section demonstrates how to set up the ESP-IDF ("ESP-IDF Framework" in "Espressif") development environment for building and running applications on ESP32 SoCs.

This project uses ESP-IDF v5.3.2 (https://docs.espressif.com/projects/esp-idf/en/v5.3.2/), locked to commit fb25eb0 (https://github.com/espressif/esp-idf/commit/fb25eb02ebcf78a78b4c34a839238a4a56accec7).

Step 1: Install Prerequisites

```
sudo apt-get install git wget flex bison gperf python3 python3-pip
python3-venv cmake ninja-build ccache libffi-dev libssl-dev dfu-util
libusb-1.0-0
```

Step 2: Get ESP-IDF

```
mkdir -p ~/esp
cd ~/esp
git clone -b release/v5.3 --recursive https://github.com/espressif/esp-
idf.git esp-idf-5.3
cd ~/esp/esp-idf-5.3
git checkout fb25eb02ebcf78a78b4c34a839238a4a56accec7
```

Step 3: Set up the Tools

ESP-IDF provides a script `install.sh` that installs the required tools such as the compiler, debugger, Python packages, etc.:

```
./install.sh all
```

Step 4: Create get_idf Alias

ESP-IDF provides a script `export.sh` that sets environment variables so the tools are usable from the command line. The script sets IDF_PATH, updates PATH with ESP-IDF tools, verifies Python compatibility, and enables idf.py auto-completion.

Create an alias for executing export.sh by adding the following line to `~/.bashrc` file:

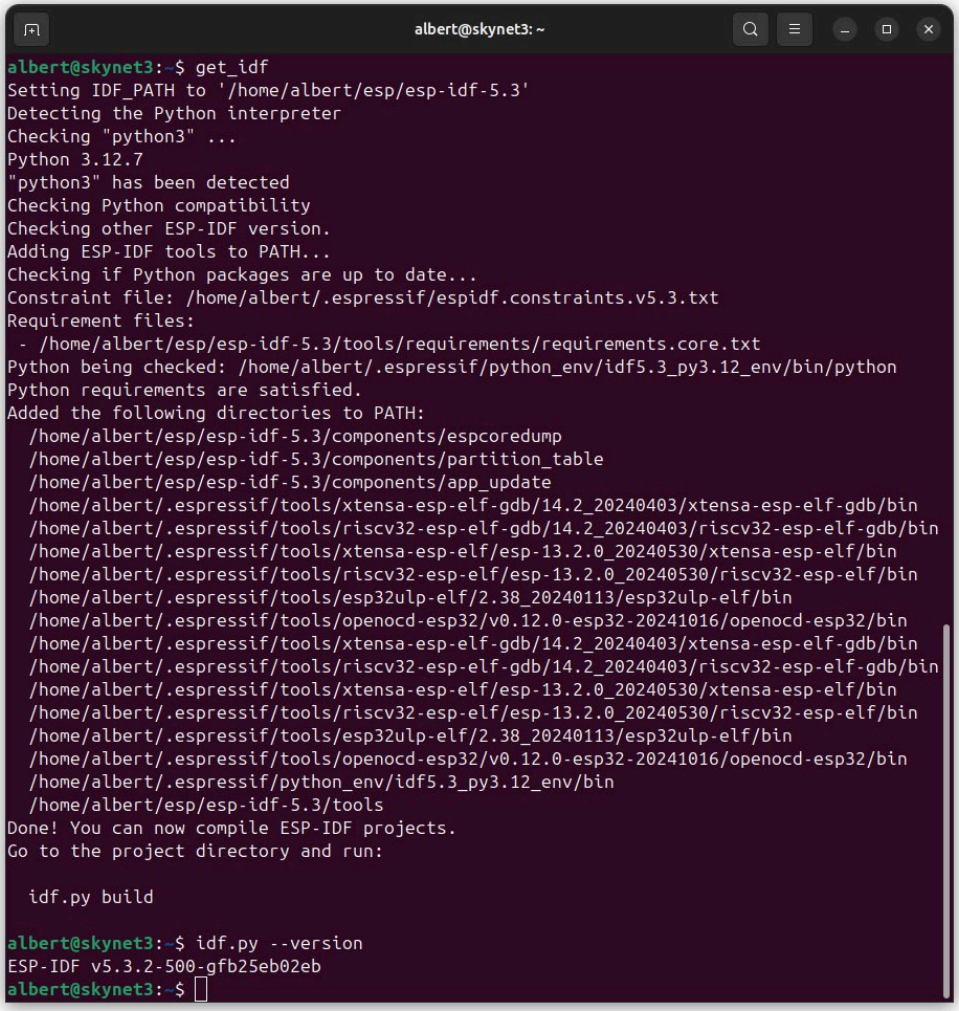
```
alias get_idf='. $HOME/esp/esp-idf-5.3/export.sh'
```

Step 5: Refresh Configuration

Restart your terminal session or run:

```
source ~/.bashrc
```

Now, running `get_idf` will set up or refresh the esp-idf environment in any terminal session.


albert@skynet3: ~
albert@skynet3:~$ get_idf
Setting IDF_PATH to '/home/albert/esp/esp-idf-5.3'
Detecting the Python interpreter
Checking "python3" ...
Python 3.12.7
"python3" has been detected
Checking Python compatibility
Checking other ESP-IDF version.
Adding ESP-IDF tools to PATH...
Checking if Python packages are up to date...
Constraint file: /home/albert/.espressif/espidf.constraints.v5.3.txt
Requirement files:
- /home/albert/esp/esp-idf-5.3/tools/requirements/requirements.core.txt
Python being checked: /home/albert/.espressif/python_env/idf5.3_py3.12_env/bin/python
Python requirements are satisfied.
Added the following directories to PATH:
/home/albert/esp/esp-idf-5.3/components/espcoredump
/home/albert/esp/esp-idf-5.3/components/partition_table
/home/albert/esp/esp-idf-5.3/components/app_update
/home/albert/.espressif/tools/xtensa-esp-elf-gdb/14.2_20240403/xtensa-esp-elf-gdb/bin
/home/albert/.espressif/tools/riscv32-esp-elf-gdb/14.2_20240403/riscv32-esp-elf-gdb/bin
/home/albert/.espressif/tools/xtensa-esp-elf/esp-13.2.0_20240530/xtensa-esp-elf/bin
/home/albert/.espressif/tools/riscv32-esp-elf/esp-13.2.0_20240530/riscv32-esp-elf/bin
/home/albert/.espressif/tools/esp32ulp-elf/2.38_20240113/esp32ulp-elf/bin
/home/albert/.espressif/tools/openocd-esp32/v0.12.0-esp32-20241016/openocd-esp32/bin
/home/albert/.espressif/tools/xtensa-esp-elf-gdb/14.2_20240403/xtensa-esp-elf-gdb/bin
/home/albert/.espressif/tools/riscv32-esp-elf-gdb/14.2_20240403/riscv32-esp-elf-gdb/bin
/home/albert/.espressif/tools/xtensa-esp-elf/esp-13.2.0_20240530/xtensa-esp-elf/bin
/home/albert/.espressif/tools/riscv32-esp-elf/esp-13.2.0_20240530/riscv32-esp-elf/bin
/home/albert/.espressif/tools/esp32ulp-elf/2.38_20240113/esp32ulp-elf/bin
/home/albert/.espressif/tools/openocd-esp32/v0.12.0-esp32-20241016/openocd-esp32/bin
/home/albert/.espressif/python_env/idf5.3_py3.12_env/bin
/home/albert/esp/esp-idf-5.3/tools
Done! You can now compile ESP-IDF projects.
Go to the project directory and run:
idf.py build
albert@skynet3:~$ idf.py --version
ESP-IDF v5.3.2-500-gfb25eb02eb
albert@skynet3:~$

ESP-Matter Setup

This section demonstrates how to set up the Espressif's SDK for Matter ("ESP-IDF Framework" in "Espressif") for building Matter applications on ESP32 SoCs.

This project uses Espressif's SDK for Matter v1.4 (https://github.com/espressif/esp-matter/tree/release/v1.4), locked to commit 30af618 (https://github.com/espressif/esp-matter/commit/30af618a6e962623a0098ad6a33b468f33dc49c7).

Step 1: Install Prerequisites

```
sudo apt-get install git gcc g++ pkg-config libssl-dev libdbus-1-dev \
    libglib2.0-dev libavahi-client-dev ninja-build python3-venv
python3-dev \
    python3-pip unzip libgirepository1.0-dev libcairo2-dev libreadline-
dev
```

Refer to the Matter Build Guide (https://github.com/espressif/connectedhomeip/blob/v1.3-branch/docs/guides/BUILDING.md) for more details.

Step 2: Clone ESP-Matter Repository

It includes esp-matter SDK and tools (e.g., CHIP-tool, CHIP-cert, ZAP).

```
cd ~/esp/
git clone -b release/v1.4 --recursive https://github.com/espressif/esp-
matter.git esp-matter-1.4
cd esp-matter-1.4
git checkout 30af618a6e962623a0098ad6a33b468f33dc49c7
```

Step 3: Bootstrap ESP-Matter

```
~/esp/esp-matter-1.4/install.sh
```

Step 4: Create get_matter Alias

The script `~/esp/esp-matter/export.sh` configures the environment. Create an alias for executing it by adding the following line to `~/.bashrc` file:

```
alias get_matter='. $HOME/esp/esp-matter/export.sh'
```

Step 5: Refresh Configuration

Restart your terminal session or run:

```
source ~/.bashrc
```

Now, running `get_matter` will set up or refresh the ESP-Matter environment in any terminal session.

References

- Connected Home over IP v1.3 (https://github.com/espressif/connectedhomeip/tree/v1.3-branch)
- Connected Home IP Documentation (https://project-chip.github.io/connectedhomeip-doc/index.html)

ESP32 Project Workflow

This section describes the workflow for creating, building, and deploying a new project.

Prerequisites:

- ESP-IDF (ESP-IDF Setup) development environment is set up.
- ESP-Matter (ESP-Matter Setup) development environment is set up.
- ESP32 ("Development Boards" in "Espressif") development board is available.
- CLion 2024.3.3 or later is installed.

Step 1: Set Up ESP-IDF Environment

Run the alias, created during the development setup, to initialise the ESP-IDF environment in the current terminal session:

```
get_idf
```

Step 2: Create a New Project

ESP-IDF provides the idf.py (https://docs.espressif.com/projects/esp-idf/en/v5.2.3/esp32/api-guides/tools/idf-py.html) command-line tool as a front-end for managing project builds, deployment, debugging, and other tasks, simplifying the workflow significantly. It integrates several essential tools, including CMake for project configuration, Ninja for building, and esptool.py for flashing the target device.

```
idf.py create-project <project name>
cd <project name>
```

Step 3: Set ESP32 as the target device

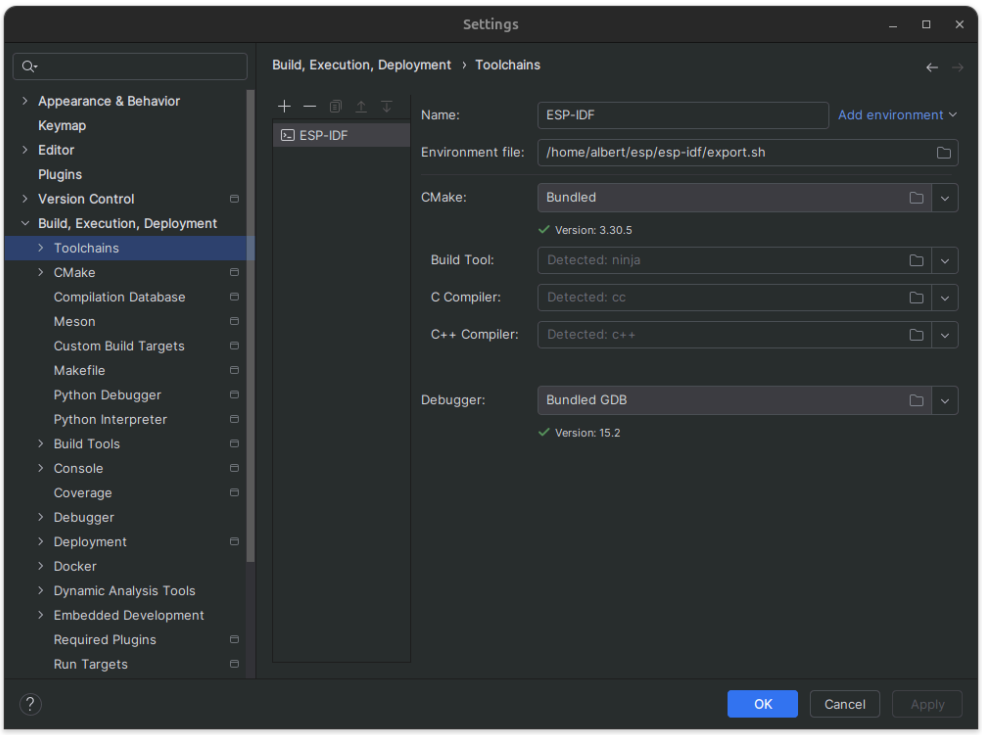
The following sets the target device:

```
idf.py set-target <chip_name>
idf.py set-target esp32h2
```

It creates a new `sdkconfig` file in the root directory of the project. This configuration file can be modified via `idf.py menuconfig`.

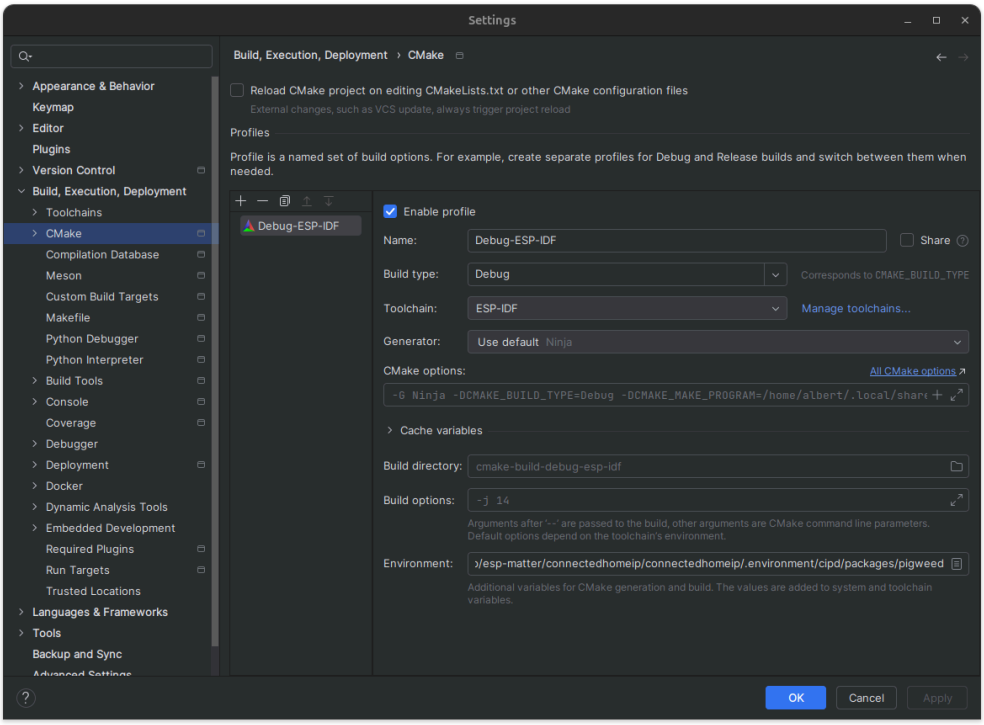
Step 4: Open the Project in CLion IDE

Create a toolchain named **ESP-IDF** with the environment file set to `esp-idf/script.sh`:

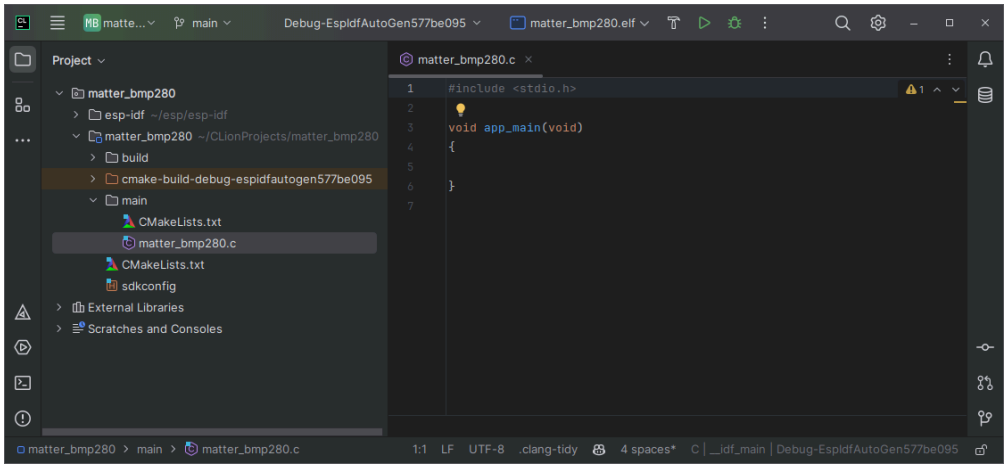


Set up a CMake profile with the following Matter environmental variables:

```
ESP_MATTER_PATH=/home/albert/esp/esp-matter-
1.4;ZAP_INSTALL_PATH=/home/albert/esp/esp-matter-
1.4/connectedhomeip/connectedhomeip/.environment/cipd/packages/zap;PATH
=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/game
s:/usr/local/games:/snap/bin:/snap/bin:/home/albert/.local/share/JetBra
ins/Toolbox/scripts:/home/albert/esp/esp-matter-
1.4/connectedhomeip/connectedhomeip/.environment/cipd/packages/pigweed
```



Structure of a new ESP-IDF project:



Step 5: Create Default Configuration

```
touch sdkconfig.defaults
```

Step 6: Update CMakeLists.txt

Update the `CMakeLists.txt` file to include the Matter SDK:

```
# For more information about build system see
# https://docs.espressif.com/projects/esp-idf/en/latest/api-
```

```
guides/build-system.html
# The following five lines of boilerplate have to be in your project's
# CMakeLists in this exact order for cmake to work correctly
cmake_minimum_required(VERSION 3.16)

# Set an error message if ESP_MATTER_PATH is not set
if(NOT DEFINED ENV{ESP_MATTER_PATH})
    message(FATAL_ERROR "Please set ESP_MATTER_PATH to the path of esp-
matter repo")
endif(NOT DEFINED ENV{ESP_MATTER_PATH})

# The set() commands should be placed after the cmake_minimum() line
but before the include() line.
set(PROJECT_VER "1.0")
set(PROJECT_VER_NUMBER 1)
set(ESP_MATTER_PATH $ENV{ESP_MATTER_PATH})
set(MATTER_SDK_PATH ${ESP_MATTER_PATH}/connectedhomeip/connectedhomeip)
set(ENV{PATH}
"$ENV{PATH}:${ESP_MATTER_PATH}/connectedhomeip/connectedhomeip/.environ
ment/cipd/packages/pigweed")

# Pulls in the rest of the CMake functionality to configure the
project, discover all the components, etc.
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
# Include common component dependencies and configurations from ESP-
Matter examples
include(${ESP_MATTER_PATH}/examples/common/cmake_common/components_incl
ude.cmake)

# Optional list of additional directories to search for components.
set(EXTRA_COMPONENT_DIRS
        "${MATTER_SDK_PATH}/config/esp32/components"
        "${ESP_MATTER_PATH}/components"
        ${extra_components_dirs_append})

# Declare the project
project(project-example)
```

```
# Set the C++ standard to C++17
idf_build_set_property(CXX_COMPILE_OPTIONS "-std=gnu++17;-Os;-
DCHIP_HAVE_CONFIG_H;-Wno-overloaded-virtual" APPEND)
idf_build_set_property(C_COMPILE_OPTIONS "-Os" APPEND)
# For RISCV chips, project_include.cmake sets -Wno-format, but does not
clear various
# flags that depend on -Wformat
idf_build_set_property(COMPILE_OPTIONS "-Wno-format-nonliteral;-Wno-
format-security" APPEND)
# Enable colored output in ninja builds
add_compile_options(-fdiagnostics-color=always -Wno-write-strings)
```

Step 7: Build the Project

```
idf.py build
```

Step 8: Determine Serial Port

Connect the ESP32 board to the computer and check under which serial port the board is visible. Serial ports have the following naming patterns: `/dev/tty`.

Step 9: Flash Project to Target

```
idf.py -p <PORT> flash
```

Step 10: Launch IDF Monitor

Use the monitor application and exit using `CTRL+]`:

```
idf.py -p <PORT> monitor
```

References

IDF Frontend - idf.py (https://docs.espressif.com/projects/esp-idf/en/stable/esp32h2/api-guides/tools/idf-py.html)

Accessory Devices

Overview

Accessory Devices are Matter (Matter)-compatible Thread End Devices ("End Device" in "Thread") that monitor and control environmental conditions within a Grow Chamber. They include **sensors** and **actuators**.

Sensors

A sensor is any device that generates some sort of output when exposed to a phenomenon.

The table below lists the sensors utilised in this project:

Sensor	Purpose
BMP280	Temperature and air pressure
BH1750 or TSL2561	Light intensity
DHT22	Temperature and humidity
MH-Z19B	Carbon dioxide (CO_2) levels

The sensor devices utilise the ESP32-H2 microcontrollers with Thread connectivity and low power consumption, enabling effective operation in various agricultural settings.

Actuators

Actuators are located on the output side of IoT solutions. They change their state based on an analog or digital signal from the microcontroller and produce an output that affects the environment.

The table below lists the actuators that will be utilised in this project:

Actuator	Purpose
Ultrasonic mist maker	Fog generation
Exhaust fan, 120mm DC	Ventilation
LED grow lights	Plant growth lighting
Air pump and air stone	Aeration for water inside Root Chamber

Actuator devices utilise the ESP32-DevKitM-1 microcontrollers. In contrast to ESP32-H2, the ESP32-DevKitM-1 does not support Thread connectivity and has higher power consumption. It is used to control actuators, which inherently consume a lot of power.

Matter Pressure and Temperature Sensor

Overview

The **Matter Pressure and Temperature Sensor** is a Matter (Matter)-compatible SED ("End Device" in "Thread") device that reads temperature and pressure data from a BMP280 sensor and exposes the readings as Matter attributes.

This section describes the development of a Matter-compatible temperature and pressure sensor.

Source Code: GitHub (https://github.com/albert-gee/matter_bmp280)

Prerequisites:

- ESP-IDF development environment (ESP-IDF Setup) is set up.
- ESP-Matter development environment (ESP-Matter Setup) is set up.
- ESP32-H2-DevKitM-1 (https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32h2/esp32-h2-devkitm-1/index.html) is available.

Hardware Assembly

Wiring BMP280 to ESP32

The **ESP32-H2** development board reads data from the **BMP280**, a low-power sensor designed for battery-powered devices. They communicate using **I2C (Inter-Integrated Circuit)**, a serial, synchronous, half-duplex protocol. The ESP32 has two I2C ports, each capable of operating as a controller or target. In this project, the ESP32-H2 acts as the controller, while the BMP280 functions as the target.

The I2C bus has two lines: the Serial Data Line (SDA) and the Serial Clock Line (SCL). On the ESP32-H2, SDA and SCL can be assigned to any available GPIO pins.

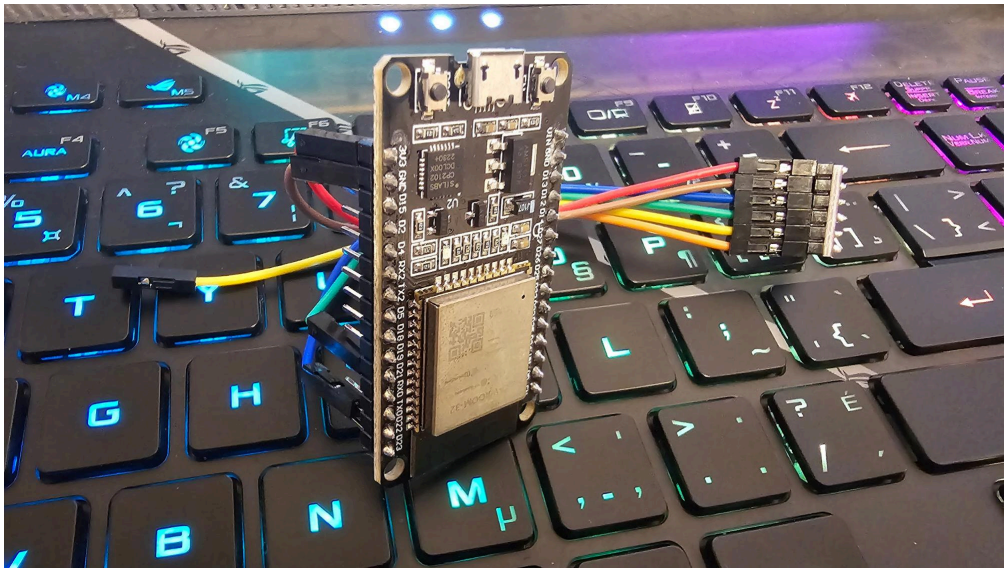
On the ESP32-DevKitM-1, GPIO21 is used for SDA, and GPIO22 is used for SCL.

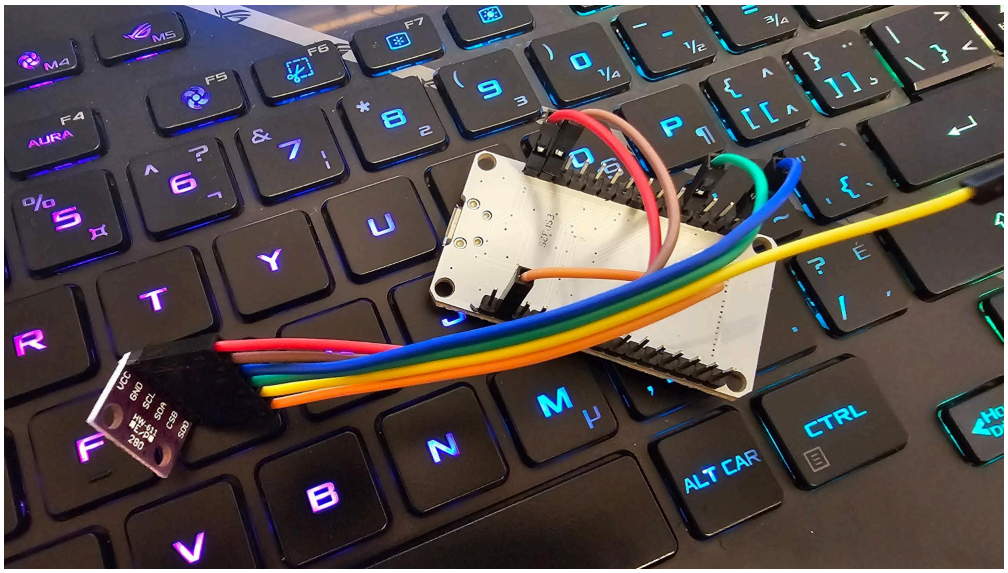
The table below shows the connections for the BMP280 sensor to the ESP32-DevKitM-1

microcontroller:

BMP280 Pin	ESP32 Pin
VCC	3.3V
GND	GND
SDA	GPIO21
SCL	GPIO22
SDO	GND

The pictures below show the connections for the BMP280 sensor to the ESP32-DevKitM-1 microcontroller:





Testing I2C Connectivity

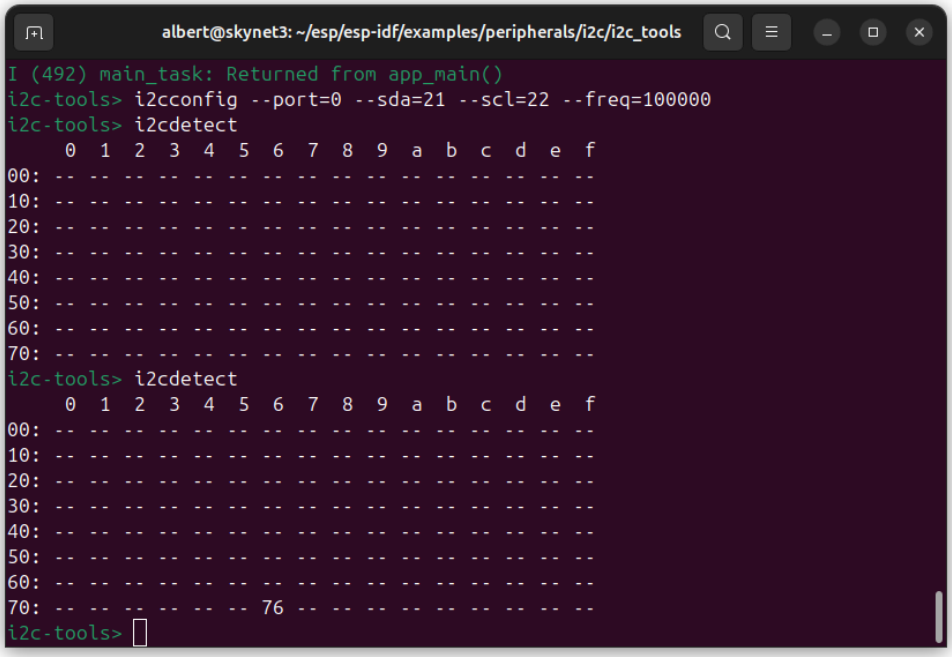
The I2C tools (https://github.com/espressif/esp-idf/tree/release/v5.4/examples/peripherals/i2c/i2c_tools) from ESP-IDF examples were used to test communication with the sensor device.

The following command configures the I2C bus with specific GPIO number, port number and frequency:

```
i2cconfig --port=0 --sda=21 --scl=22 --freq=100000
```

The following command scans an I2C bus for devices and output a table with the list of detected devices on the bus:

```
i2cdetect
```



It displays the address 0x76 since the SDO pin is connected to GND.

The following command get the value of the “ID” register which contains the chip identification number chip_id, which is 0x58:

```
i2cget -c 0x76 -r 0xD0 -l 1
```

- -c option to specify the address of I2C device (acquired from i2cdetect command).
- -r option to specify the register address you want to inspect.
- -l option to specify the length of the content.

Firmware Development

The firmware for the ESP32-H2 was developed using the **ESP-IDF framework**.

The BMP280 datasheet (*BST-BMP280-DS001-26*) is used as a reference during development.

Project Bootstrap

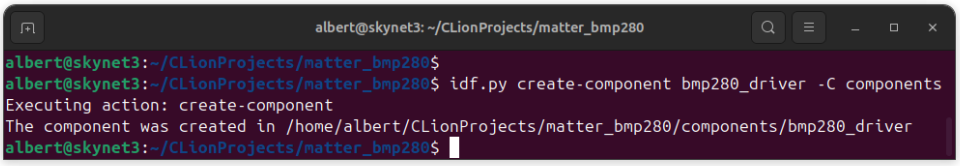
Follow the following steps from the ESP32 Project Workflow (ESP32 Project Workflow) guide:

- Step 1: Set Up ESP-IDF Environment ("Step 1: Set Up ESP-IDF Environment" in "ESP32 Project Workflow")
- Step 2: Create a New Project ("Step 2: Create a New Project" in "ESP32 Project Workflow")
- Step 3: Set ESP32 as the target device ("Step 3: Set ESP32 as the target device" in "ESP32 Project Workflow")
- Step 4: Open the Project in CLion IDE ("Step 4: Open the Project in CLion IDE" in "ESP32 Project Workflow")
- Step 5: Create Default Configuration ("Step 5: Create Default Configuration" in "ESP32 Project Workflow")
- Step 6: Update CMakeLists.txt ("Step 6: Update CMakeLists.txt" in "ESP32 Project Workflow")

Driver Component Development

The command below creates a new component named `bmp280_driver` inside the `components` directory.

```
idf.py create-component bmp280_driver -C components
```



The ESP-IDF framework includes a driver for working with I2C devices. To ensure that this component can use the driver, the `REQUIRES` directive must be added to `components/bmp280_driver/CMakeLists.txt` as follows:

```
REQUIRES "driver"
```

This directive ensures that the build system includes the I2C driver as a dependency for the component during the build process.

The `i2c_utils.h` header file declares utility functions for initializing, managing, and performing operations on an I2C master bus and its connected devices.

The `i2c_utils.c` file implements these functions and includes `driver/i2c_master.h` ($IDF_PATH/components/driver/i2c/include/driver/i2c_master.h) to access the driver's API in controller mode.

The BMP280 code is designed according to the specifications outlined in the BST-BMP280-DS001-26 datasheet. Communication with the sensor is performed through read and write operations on its 8-bit registers.

Register Name	Address	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
temp_xlsb	0xFC	temp_xlsb<7:4>				0	0	0	0	0x00
temp_lsb	0xFB	temp_lsb<7:0>								0x00
temp_msb	0xFA	temp_msb<7:0>								0x80
press_xlsb	0xF9	press_xlsb<7:4>				0	0	0	0	0x00
press_lsb	0xF8	press_lsb<7:0>								0x00
press_msb	0xF7	press_msb<7:0>								0x80
config	0xF5	t_sb[2:0]			filter[2:0]				spi3w_en[0]	0x00
ctrl_meas	0xF4	osrs_t[2:0]			osrs_p[2:0]			mode[1:0]		0x00
status	0xF3					measuring[0]			im_update[0]	0x00
reset	0xE0	reset[7:0]								0x00
id	0xD0	chip_id[7:0]								0x58
calib25...calib00	0xA1…0x88	calibration data								individual

Registers:	Reserved registers	Calibration data	Control registers	Data registers	Status registers	Revision	Reset
Type:	do not write	read only	read / write	read only	read only	read only	write only

The `bmp280_driver.h` header file declares the configurations, constants, and functions needed to interface with and operate the sensor, based on the datasheet.

The `bmp280_driver.c` file implements these functions, providing support for sensor initialization, configuration, and data handling. It adheres to the datasheet to verify the sensor, apply compensation to raw measurements, and configure parameters like oversampling, operating modes, and filters.

The BMP280 sensor uses factory-programmed calibration parameters to adjust raw data for accurate measurements. These parameters are stored in the sensor's non-volatile memory (NVM) during production, are unique to each device, and cannot be modified by the user, as described in Section 3.11.2 of the datasheet. The `bmp280_calibration.h` header declares the constants, structures, and functions needed to handle this calibration data. It includes functionality to read the parameters and apply them to raw

pressure and temperature readings. The `bmp280_calibration.c` file implements these functions, following the temperature and pressure compensation formulas specified in Section 3.11 of the datasheet.

Matter Component Development

The command below creates a new component named bmp280_driver inside the components directory.

```
idf.py create-component matter_interface -C components
```

The `matter_temperature_sensor.h` file defines functions for managing a Matter-compatible temperature sensor.

The `matter_temperature_sensor.c` file implements these functions using Espressif's Matter SDK, built on the Matter SDK.

In the Matter ecosystem, devices are represented as Nodes, which consist of Endpoints representing specific functions, such as temperature measurement. Endpoints are organized into Clusters that group related features. This implementation sets up a Node with an Endpoint for temperature measurement, configures its Clusters, and manages communication within the Matter network.

Integration of BMP280 Driver with ESP-Matter

The bmp280_driver component outputs temperature and pressure readings as 32-bit signed integers (int32_t), following the BMP280 datasheet specifications. These integers are used to process the sensor's raw 20-bit data for precise calculations, including compensation and scaling. The readings represent scaled real-world values to preserve decimal precision using integer arithmetic. For example, a temperature of 25.25°C is stored as 2525 (in units of 0.01°C), and a pressure of 1013.25 is stored as 101325 (in Pascals).

During integration with ESP-Matter, an issue arose because ESP-Matter defines attributes like measured_value, min_measured_value, and max_measured_value in its Temperature and Pressure Measurement Clusters using the nullable< int16_t> data type. This required the BMP280's outputs, represented as int32_t, to be reduced to fit within the int16_t range of -32,768 to 32,767. The BMP280 outputs temperature values within a range of -40°C to +85°C and pressure values within a range of 300 hPa to 1100 hPa. For temperature, no additional scaling was necessary because even the maximum value,

when scaled (e.g., 85.00°C becomes 8500), fits comfortably within the int16_t range without any loss of precision.

For pressure, however, scaled values can exceed the limit of the int16_t range. Multiplying higher values by 100 to preserve decimal precision increases their magnitude beyond the int16_t limit. For example, a pressure of 1013.25 becomes 101325 when scaled, which far exceeds the int16_t range.

To address this, the scaling approach was modified. Instead of multiplying by 100 to represent hundredths of Pascals, pressure values were scaled by 10. This reduced the magnitude of the scaled values, allowing them to fit within the int16_t range. For example, a pressure of 1013.25 is scaled to 10132, which fits comfortably within the range. Similarly, the maximum pressure of 1100.00 is scaled to 11000.

Reading the Temperature

To manage temperature readings, two approaches were considered:

- Regularly poll the temperature sensor, for example, using a timer. Each new reading is written directly to the attribute's stored value.
- Update the temperature reading only when a client requests it. In this approach, the sensor retrieves the latest temperature data and writes it to the attribute before responding to the client.

The second approach requires a mechanism to dynamically fetch the latest sensor data during a client read operation. ESP-Matter provides native support for callbacks during attribute write operations, but not for reads. In the ESP-Matter repository on GitHub, Issue #264 (https://github.com/espressif/esp-matter/issues/264) includes a discussion about using the `ATTRIBUTE_FLAG_OVERRIDE` flag to implement such functionality. The discussion describes a workaround involving modifications to the framework's internal `esp_matter_attribute.cpp` file to override the default attribute behavior and introduce custom read logic.

A more maintainable alternative would be to leverage the flag directly within application code. This approach avoids modifying the framework itself. However, it introduces additional complexity and can cause delays in client responses, as the sensor data must be fetched in real-time for each read request. Due to these challenges, the first option was chosen for its simplicity and faster response times.

Testing

Testing with CHIP-Tool

Refer to CHIP-Tool (CHIP-Tool) for more details.

```
matter onboardingcodes ble
```

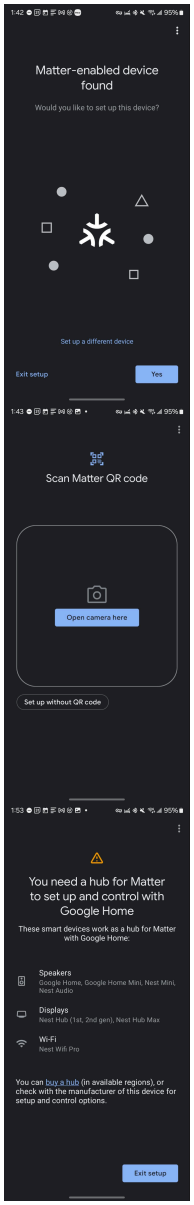
```
matter esp attribute get 0x0001 0x00000402 0x00000000
matter esp attribute get 0x0002 0x00000403 0x00000000
```

Connecting to Google Home

It is also possible to use third-party software, such as the Google Home app, as a controller. However, the device appears offline in Google Home despite being successfully commissioned. To address this issue, a structured troubleshooting process was undertaken:

- Flash Memory Reset: The device's flash memory was erased to remove any residual configurations that could interfere with its functionality.
- Network Configuration Check: It was confirmed that both the mobile device and the hardware were connected to a 2.4 GHz Wi-Fi network, as the system does not support 5 GHz networks.
- Device Reset: The hardware was reset to ensure proper initialization and resolve any temporary issues.

Vaishali Avhale, Associate QA Engineer at Espressif Systems, provided feedback on a related issue in the Espressif ESP-Matter GitHub repository (Issue #1125 (https://github.com/espressif/esp-matter/issues/1125)). She tested the system using a Google Nest Hub 2nd Generation as a hub and confirmed that it worked correctly in this configuration. She noted that a hub device is a required component for proper operation within Google's ecosystem.



References

- BMP280 – Data sheet (https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bmp280-ds001-26.pdf)
- Telink Matter Developers Guide (https://wiki.telink-semi.cn/doc/an/TelinkMatterDevelopersGuide_en.pdf)

- ESP-Matter Sensors Example (https://github.com/espressif/esp-matter/tree/main/examples/sensors)

Matter Relay Switch

Overview

The **Matter Relay Switch** is a Matter (Matter)-compatible REED ("End Device" in "Thread") device that can be controlled remotely to turn a relay on or off. It can be used to control various devices, such as lights or fans.

This section describes the development of the **Matter Relay Switch**.

Source Code: GitHub (https://github.com/albert-gee/matter_relay)

Prerequisites:

- ESP-IDF development environment (ESP-IDF Setup) is set up.
- Matter development environment (Matter Interface) is set up.
- ESP32-H2-DevKitM-1 (https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32h2/esp32-h2-devkitm-1/index.html) is available.

Hardware Assembly

Wiring Relay to ESP32

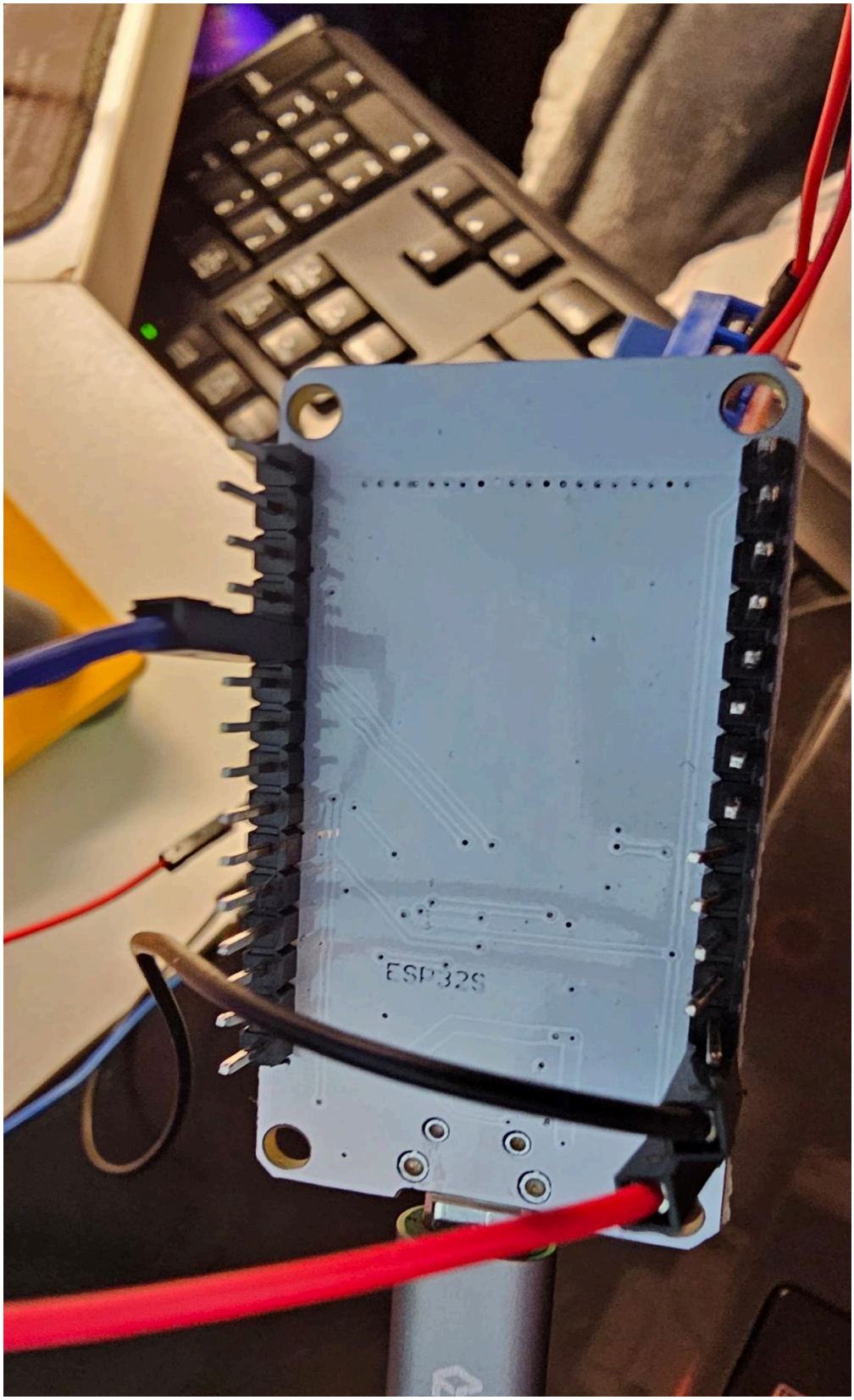
The **Valefod 5V 1-Channel Relay Module** is used to control loads up to 10A using a low-power control signal from a microcontroller such as an Arduino or ESP32. It features optocoupler isolation for improved signal integrity and supports both high and low trigger configurations.

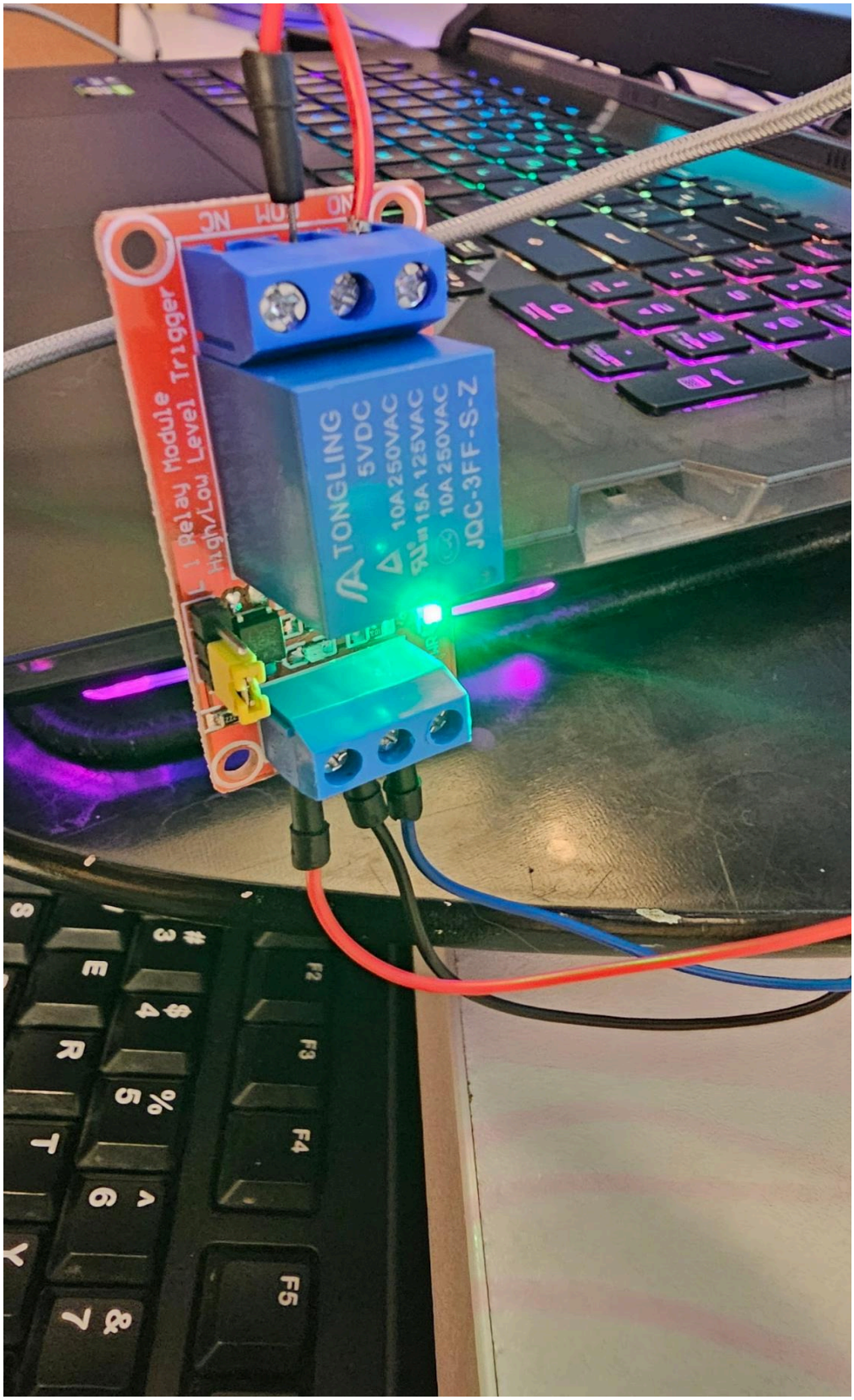
The relay has the following terminal layout:

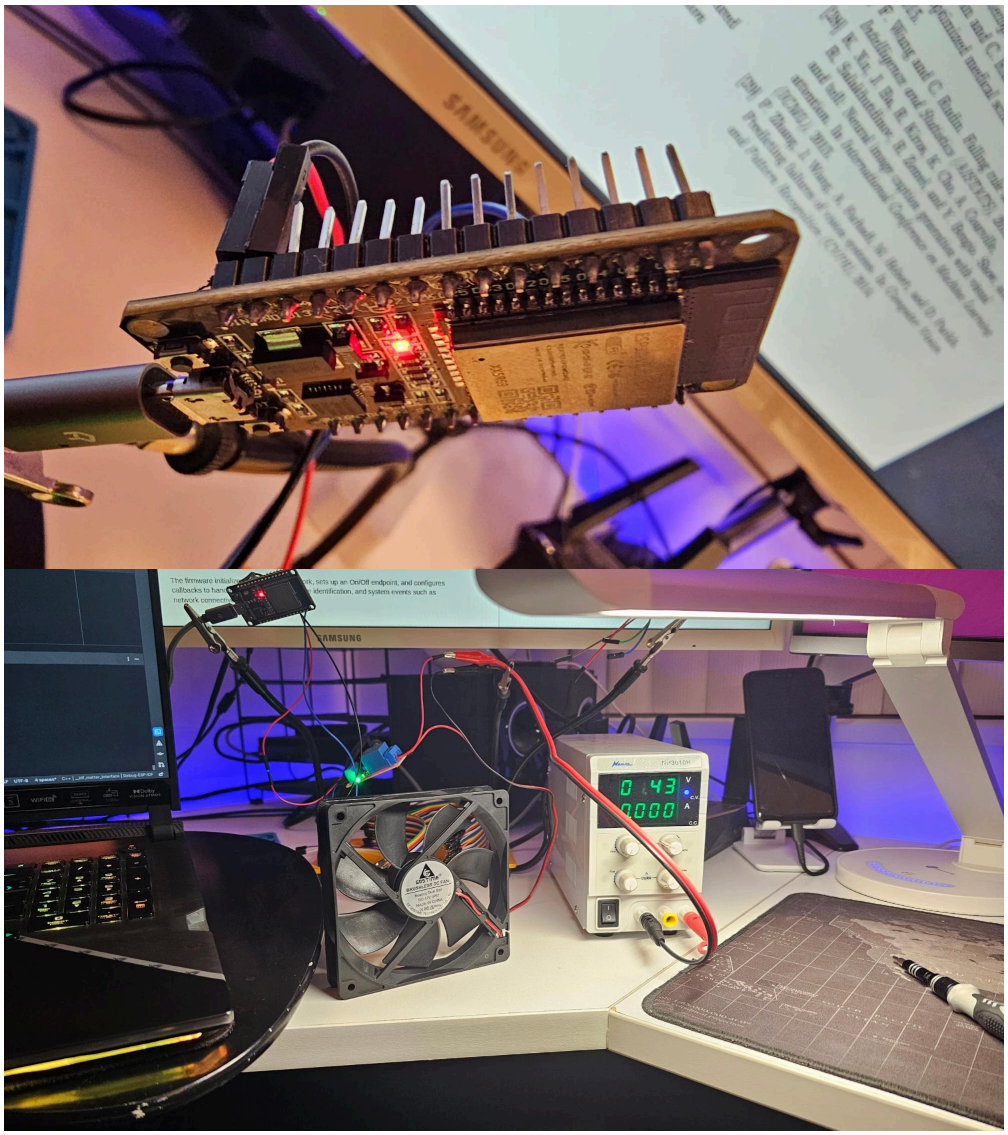
- **Side 1: Control Terminals**
  - DC+ (Power Supply Positive): Connect to a 5V power supply.
  - DC- (Power Supply Negative): Connect to the ground of the power supply.
  - IN (Input Signal): Connect to a GPIO pin on the microcontroller to control the relay.
- **Side 2: Switch Terminals**

- COM (Common Terminal): Shared terminal for the relay's switch.
- NO (Normally Open): Open circuit by default; closes when the relay is activated.
- NC (Normally Closed): Closed circuit by default; opens when the relay is activated.

The Normally Open (NO) terminal is used when the circuit should remain open (off) by default, requiring activation to close the circuit. The Normally Closed (NC) terminal is used when the circuit should remain closed (on) by default, opening only when the relay is activated.


ESP32S


1 Relay Module
High/Low Level Trigger
NC
NO
TONGLING
5VDC
10A 250VAC
15A 125VAC
10A 250VAC
JQC-3FF-S-Z



Firmware Development

The firmware uses the ESP-IDF ("ESP-IDF Framework" in "Espressif") framework with the SDK for Matter. It sets up an On/Off endpoint and configures callbacks to manage Matter attributes, enabling GPIO-based relay control.

Project Bootstrap

Follow the following steps from the ESP32 Project Workflow (ESP32 Project Workflow) guide:

- Step 1: Set Up ESP-IDF Environment ("Step 1: Set Up ESP-IDF Environment" in "ESP32 Project Workflow")

- Step 2: Create a New Project ("Step 2: Create a New Project" in "ESP32 Project Workflow")
- Step 3: Set ESP32 as the target device ("Step 3: Set ESP32 as the target device" in "ESP32 Project Workflow")
- Step 4: Open the Project in CLion IDE ("Step 4: Open the Project in CLion IDE" in "ESP32 Project Workflow")
- Step 5: Create Default Configuration ("Step 5: Create Default Configuration" in "ESP32 Project Workflow")
- Step 6: Update CMakeLists.txt ("Step 6: Update CMakeLists.txt" in "ESP32 Project Workflow")

GPIO Configuration

The header file `$IDF_PATH/components/esp_driver_gpio/include/driver/gpio.h` can be included with:

```
#include "driver/gpio.h"
```

This header file is a part of the API provided by the `esp_driver_gpio` component. To declare that your component depends on esp_driver_gpio, the following line has to be added to CMakeLists.txt:

```
REQUIRES esp_driver_gpio
```

or

```
PRIV_REQUIRES esp_driver_gpio
```

Testing

Testing with CHIP-Tool

Refer to CHIP-Tool (CHIP-Tool) for more details.

```
matter onboardingcodes ble
```

```
matter esp attribute set 0x1 0x6 0x0 1
matter esp attribute get 0x1 0x6 0x0
```

Thread Border Routers

Overview

Thread Border Routers ("Border Router" in "Thread") provide IP connectivity between the Thread Network of accessory devices (Accessory Devices) and adjacent external W-Fi network, such as a home LAN, building network, or the broader Internet.

This section walks through how to set up and run ESP Basic Thread Border Router (ESP Basic Thread Border Router) and ESP Matter Thread Border Router (ESP Matter Thread Border Router) examples on the ESP Thread Border Router ("ESP Thread Border Router Solution" in "Espressif") board.

ESP Basic Thread Border Router

Overview

The `ESP Basic Thread Border Router` example from ESP_IDF ("ESP-IDF Framework" in "Espressif") demonstrates how to build firmware for a basic Thread Border Router device running on Espressif's hardware.

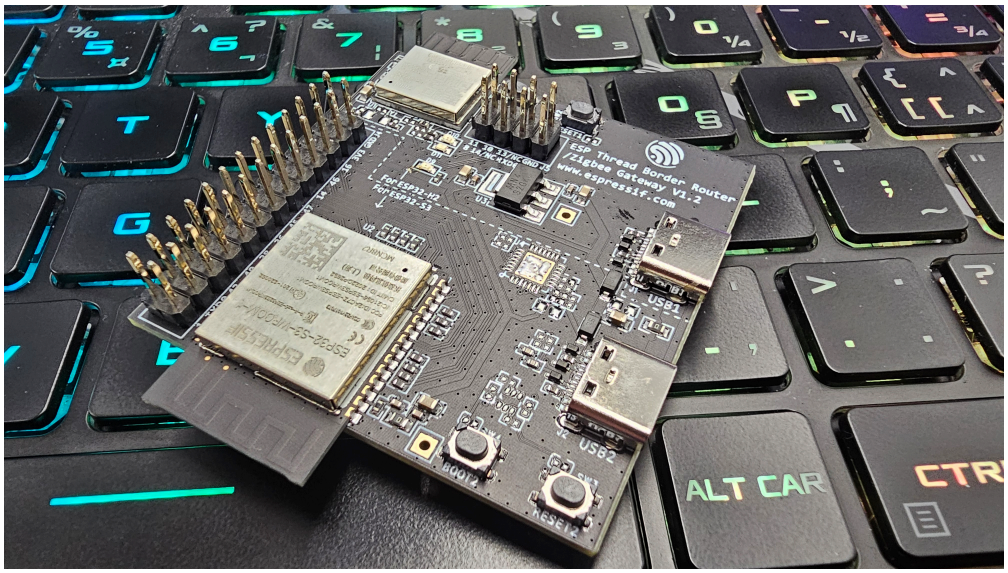
The firmware configures the OpenThread platform using default radio, host, and port settings. Optionally, it enables external coexistence and sets up mDNS (with the hostname "esp-ot-br"), along with OTA and web server support if enabled. Finally, it launches the Border Router.

Build and Run

This section demonstrates how to build and run the **ESP Basic Thread Border Router** example.

Prerequisites:

- ESP-IDF development environment (ESP-IDF Setup) is set up.
- ESP THREAD BR-ZIGBEE GW ("Border Router" in "Thread") board



Step 1: Build the RCP Image

```
get_idf
cd $IDF_PATH/examples/openthread/ot_rcp/
idf.py set-target esp32h2   # Select the ESP32-H2. Skipping this step
would result in a build error.
idf.py build
```

The firmware does not need to be manually flashed onto the device. It will be integrated into the Border Router firmware and automatically installed onto the ESP32-H2 chip during the first boot-up. `idf.py menuconfig` can be used for customized settings.

Step 2: Clone Espressif Thread Border Router SDK

This project uses Espressif Thread Border Router SDK locked to commit cf3a09f (https://github.com/espressif/esp-thread-br/commit/cf3a09f5f44991a4e65b2d1c5113637e1d086b68).

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-thread-br.git
cd ~/esp/esp-thread-br/
git checkout cf3a09f5f44991a4e65b2d1c5113637e1d086b68
cd ~/esp/esp-thread-br/examples/basic_thread_border_router
```

Step 3: Configure the Device (Optional)

This section describes how to configure the device using `idf.py menuconfig`. This step is *optional* because the device can be configured using the OpenThread CLI ("OpenThread CLI" in "Thread") after flashing the firmware.

The default configuration in `sdkconfig.default` file is designed to work out of the box on the ESP Thread Border Router board with ESP32-S3 as the default SoC target. For other SoCs, the target must be configured using `idf.py set-target <chip_name>`.

The command below opens the configuration menu:

```
idf.py menuconfig
```

Enable **automatic start mode** and the **Web GUI**:

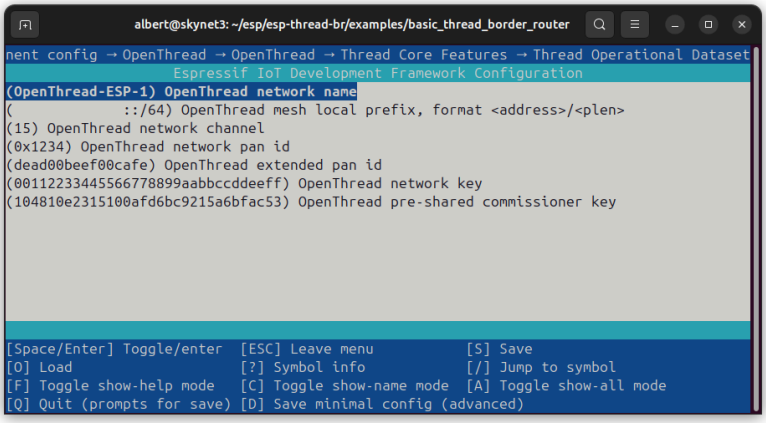
- ESP Thread Border Router Example → Enable the automatic start mode in Thread Border

In **automatic start mode**, the device first attempts to use the Wi-Fi SSID and password stored in **NVS** (Non-Volatile Storage). If no Wi-Fi credentials are found in NVS, it uses **EXAMPLE_WIFI_SSID** and **EXAMPLE_WIFI_PASSWORD**, retrieved from the following configuration options:

- Example Connection Configuration → WiFi SSID
- Example Connection Configuration → WiFi Password

Additionally, **Thread dataset** can be configured:

Component config → OpenThread → Thread Core Features → Thread Operational Dataset



Step 4: Connect the ESP Thread Border Router Board

Use **USB2 (ESP32-S3)** on the ESP Thread Border Router Board to connect the board to the computer. Only the ESP32-S3 (main SoC) port needs to be connected. The main SoC automatically programs the Thread co-processor.

Step 5: Build and Flash

```
idf.py build
idf.py -p <PORT> flash monitor
```

OpenThread CLI

This section demonstrates how to use the OpenThread CLI ("OpenThread CLI" in "Thread") on the ESP Thread Border Router.

OpenThread CLI is enabled in the ESP OpenThread component (https://github.com/espressif/esp-idf/blob/master/components/openthread/Kconfig) by default. It can also be enabled/disabled using the `idf.py menuconfig` command:

- `Component config → OpenThread → Thread Console → Enable OpenThread Command-Line Interface`

The `ESP Thread Border Router` project extends the standard `OpenThread CLI` by including the OpenThread Extension Commands (https://github.com/espressif/esp-thread-br/tree/main/components/esp_ot_cli_extension#wifi) component with additional commands demonstrated below, such as Wi-Fi management and IP address printing.

Wi-Fi Management

Connect the ESP Thread Border Router to a Wi-Fi network:

```
wifi connect -s <SSID> -p <PASSWORD>
```

Disconnect the ESP Thread Border Router from the Wi-Fi network:

```
wifi disconnect
```

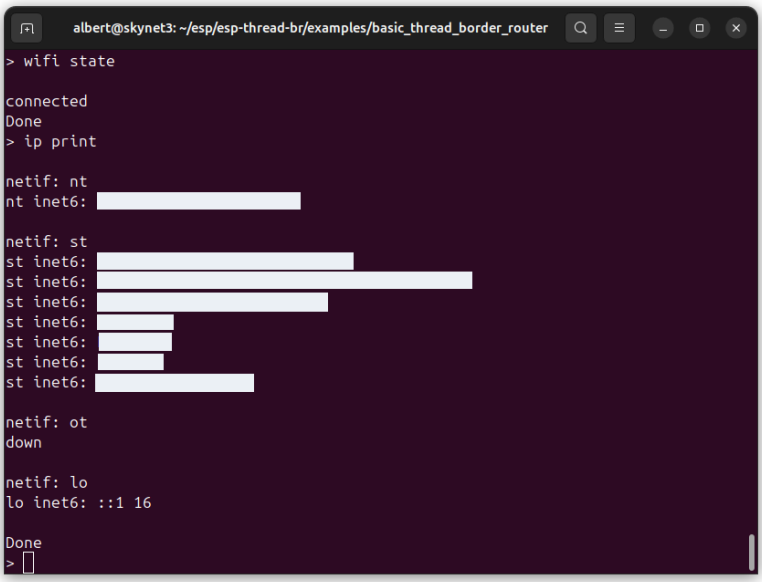
Print the current Wi-Fi network state:

```
wifi state
```

IP Addresses and Network Interfaces

Print all the IP address on each interface of lwIP of the Thread Border Router:

```
ip print
```



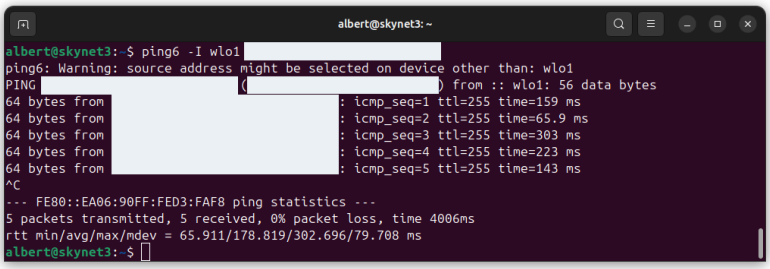
The output lists network interfaces (netif):

- **nt**
- **ot ** (OpenThread Network Interface (https://github.com/espressif/esp-idf/blob/release/v5.3/components/openthread/src/esp_openthread_lwip_netif.c#L136)),
- **st ** (Wi-Fi Station (STA) interface (https://github.com/espressif/esp-idf/blob/release/v5.3/components/esp_netif/lwip/netif/wlanif.c#L223)), this includes the **Global Unicast Address** (**GUA**), which falls within the 2000::/3 range (starts with 2xxx, 3xxx). This address is globally routable and can be used for external access if the Wi-Fi router's firewall allows incoming connections to this address.
- **lo** (Loopback interface (https://github.com/espressif/esp-lwip/blob/2.1.3-esp/src/core/netif.c#L151)).

If the Thread Border Router is connected to a Wi-Fi network, we can ping it from another device. The following commands should be run on a Linux machine to find the name of the Wi-Fi network interfaces and then use it to ping the Thread Border Router:

```
ip addr
ping6 -I <INTERFACE> <DEVICE_IP>
```

```
ping6 -I wlo1 fe80::aaaa:0000:0000:0000
ping6 -I wlo1 2604::aaaa:0000:0000:0000
```



Forming Thread network

The following commands form a Thread network (Forming a Network):

1. *Optional*. Delete any previous settings stored on non-volatile memory, and then trigger a platform reset:

   ```
   factoryreset
   ```

2. Initialize a new dataset:

   ```
   dataset init new
   ```

3. Commit the new Operational Dataset buffer to Active Operational Dataset:

   ```
   dataset commit active
   ```

4. Enable IPv6 communication:

   ```
   ifconfig up
   ```

5. Enable Thread interface

   ```
   thread start
   ```

6. Show the Active Operational Dataset. The optional `-x` argument prints it as hex-encoded TLVs:

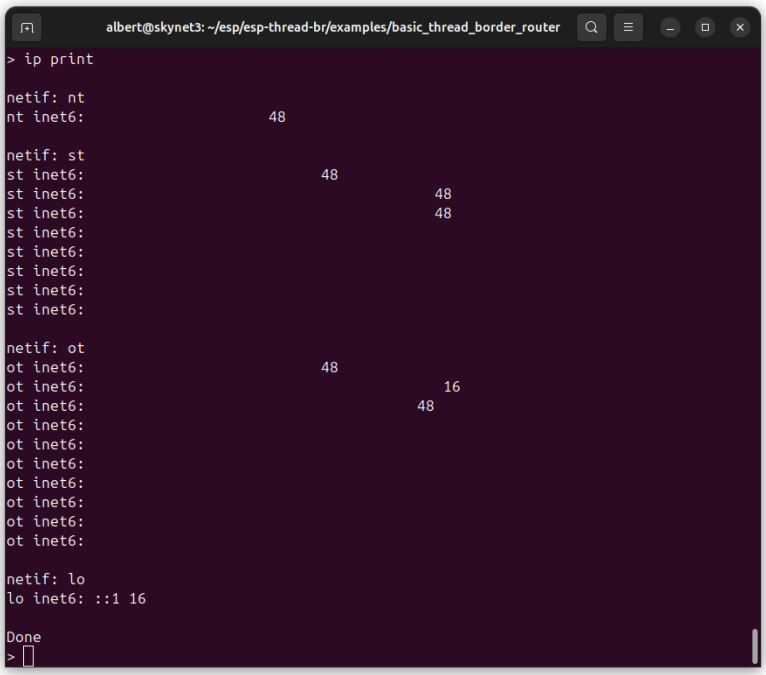
```
dataset active -x
```

```
> dataset active

Active Timestamp: 1
Channel: 23
Wake-up Channel: 21
Channel Mask: 0x07fff800
Ext PAN ID: cee2281bfad406a1
Mesh Local Prefix:
Network Key: aba4f103d160b985d67d746026503b96
Network Name: OpenThread-91dd
PAN ID: 0x91dd
PSKc: dadef003379811fafe86f995dcdfd127
Security Policy: 672 onrc 0
Done
> dataset active -x

0e080000000000010000000300000174a0300001535060004
11fafe86f995dcdfd1270c0402a0f7f8
Done
```

The ip print ("IP Addresses and Network Interfaces" in "ESP Basic Thread Border Router") command now shows the Thread interface with the IPv6 address:



Managing Border Routing Manager

The Border Routing Manager starts automatically after enabling the Thread interface. The `br` command can be used to manage the Border Routing Manager.

Print the current state of Border Routing Manager:

```
br state
```

Initializes the Border Routing Manager:

```
br init <infrastructure-network-index> <is-running>
```

Enables the Border Routing Manager:

```
br enable
```

Information Commands

The `platform` command prints the current platform. The `version` and `rcp version` print the OpenThread version and the radio version respectively.

RESTful API and Web GUI

The ESP Thread Border Router provides a RESTful API and Web GUI for managing the Thread Border Router. It is disabled by default and can be enabled using `idf.py menuconfig`:

- `ESP Thread Border Router Example → Enable the web server in Thread Border Router`

Web GUI Access

The Web GUI can be accessed using the IP address of the Thread Border Router:

```
http://<DEVICE_IP>/index.html
```

The Web GUI can be accessed using the IPv4 address assigned by the Wi-Fi router or the device's IPv6 Global Unicast Address (see `ip print` in extension-commands). IPv4 is only accessible within the local network unless port forwarding is configured on the router. IPv6 can be accessed externally if a firewall rule is added on the router.

RESTful API Client Library Generation

The API is documented using the OpenAPI specification in the openapi.yaml (https://github.com/espressif/esp-thread-br/blob/main/components/esp_ot_br_server/src/openapi.yaml) file. OpenAPI Generator (https://github.com/OpenAPITools/openapi-generator) can be used to generate a client library.

The following commands should be run on a Linux machine with installed Docker and Node.js to generate a TypeScript client library for the ESP-Thread Border Router server:

```
docker run --rm -v "${PWD}:/local" openapitools/openapi-generator-cli
generate -i /local/openapi.yaml -g typescript-fetch -o
/local/out/typescript-fetch
sudo chown -R $USER:$USER out/
npm install --safe-dev typescript
```

References

- Espressif Thread Border Router SDK (https://github.com/espressif/esp-thread-br)

- OpenThread - ESP Thread Border Router (https://mattercoder.com/codelabs/how-to-install-border-router-on-esp32/?index=..%2F..index#0)
- Espressif - Build and Run ESP Thread Border Router (https://docs.espressif.com/projects/esp-thread-br/en/latest/dev-guide/build_and_run.html)
- OpenThread - Build a Thread Network with the ESP32H2 and ESP Thread Border Router Board (https://openthread.io/codelabs/esp-openthread-hardware)
- How to Install Border Router on ESP32-DevKit and ESP32-H2 (https://mattercoder.com/codelabs/how-to-install-border-router-on-esp32/?index=..%2F..index#0)
- Web GUI (https://docs.espressif.com/projects/esp-thread-br/en/latest/codelab/web-gui.html)
- Global Unicast Address (https://docs.oracle.com/cd/E19120-01/open.solaris/819-3000/ipv6-overview-130/index.html)

ESP Matter Thread Border Router

Overview

The `ESP Matter Thread Border Router` example from Espressif's SDK for Matter ("ESP Matter Solution" in "Espressif") demonstrates how to build firmware for a Matter-compliant Thread Border Router device running on Espressif's hardware.

The firmware creates a Matter node and configures a Thread Border Router endpoint. If Thread support is enabled, it configures the OpenThread platform with default radio, host, and port settings. Finally, it starts the Matter stack with an event callback to handle network events and initializes the diagnostic console if enabled.

When the ESP32 connects to Wi-Fi and gets an IP, it sets up Wi-Fi as the backbone for OpenThread and starts the Thread Border Router to enable communication between Thread and Wi-Fi. A static variable stops it from running more than once.

Build and Run

This section demonstrates how to build and run the `ESP Matter Thread Border Router` example from Espressif's SDK for Matter ("ESP Matter Solution" in "Espressif").

Prerequisites:

- ESP-IDF development environment (ESP-IDF Setup) is set up.
- ESP-Matter development environment (ESP-Matter Setup) is set up.
- ESP THREAD BR-ZIGBEE GW ("Border Router" in "Thread") board

Step 1: Build and Configure the RCP Image

The build process is similar to the Basic Thread Border Router example ("Build and Run" in "ESP Basic Thread Border Router"). The following steps from the Basic Thread Border Router example are required if not already completed:

1. Follow Step 1: Build the RCP Image ("Step 1: Build the RCP Image" in "ESP Basic Thread Border Router").

2. Follow optional Step 3: Configure the Device ("Step 3: Configure the Device (Optional)" in "ESP Basic Thread Border Router").

Step 2: Set Up Environment

Set up the environment and navigate to the `ESP Matter Thread Border Router` example directory:

```
get_idf
get_matter
cd $ESP_MATTER_PATH/examples/thread_border_router
```

Step 3: Set target to ESP32-S3

```
idf.py set-target esp32s3
```

Step 4: Connect the ESP Thread Border Router Board

Follow Step 4: Connect the ESP Thread Border Router Board ("Step 4: Connect the ESP Thread Border Router Board" in "ESP Basic Thread Border Router") from the Basic Thread Border Router example.

Step 5: Build and Flash

```
idf.py build
idf.py -p <PORT> flash monitor
```

Matter CLI

This section demonstrates how to use the Matter CLI on the ESP Matter Thread Border Router.

The device automatically starts BLE advertising when powered on. The following command can be used to check the state:

```
matter ble adv state
```

The following command prints the Matter configuration, including the PIN code and

Discriminator, needed for commissioning:

```
matter config
```

CHIP-Tool

The Thread Border Router management cluster allows provisioning of the Thread interface using a Matter commissioner.

Commissioning to Wi-Fi over BLE

Refer to Commissioning to Wi-Fi over BLE using CHIP Tool ("Commissioning to Wi-Fi over BLE" in "CHIP-Tool").

Joining a Thread Network

After commissioning, a fail-safe timer must be armed:

```
./chip-tool generalcommissioning arm-fail-safe <TIMEOUT_IN_SEC> 1
<NODE_ID> 0
./chip-tool generalcommissioning arm-fail-safe 120 1 0x1122 0
```

Provision the Border Router using an active dataset in HEX TLV format (the same format used for commissioning a Matter over Thread ("Commissioning to Thread over BLE" in "CHIP-Tool") device via the `ble-thread` command). The **Thread Border Router Management Cluster** should be used:

```
./chip-tool threadborderroutermanagement set-active-dataset-request
hex:<ACTIVE_DATASET> <NODE_ID> 1
```

If the active dataset command succeeds, complete the commissioning process by disarming the fail-safe timer and committing the configuration to non-volatile storage:

```
./chip-tool generalcommissioning commissioning-complete <NODE_ID> 1
```

Verify the configuration by retrieving the active dataset:

```
./chip-tool threadborderroutermanagement get-active-dataset-request
```

```
<NODE_ID> 1
```

The response includes a `DatasetResponse` containing the active dataset in hex-encoded format.

References

- ESP-Matter Thread Border Router (https://github.com/espressif/esp-matter/tree/main/examples/thread_border_router)
- CHIP-Tool - Thread Border Router Usage (https://project-chip.github.io/connectedhomeip-doc/platforms/nxp/nxp_otbr_guide.html)

System Control

Overview

This chapter describes the tools and interfaces for configuring, managing, and monitoring the system. It demonstrates existing solutions - like the ESP OpenThread CLI (ESP OpenThread CLI) for Thread management and the CHIP-Tool (CHIP-Tool) for Matter - and introduces the new Orchestrator (Orchestrator) device and Control Panel (Control Panel).

ESP OpenThread CLI

This section describes how to build and run the OpenThread CLI ("OpenThread CLI" in "Thread") example project from the ESP-IDF ("ESP-IDF Framework" in "Espressif"). It can be used to test and debug the OpenThread stack on ESP32 series SoCs ("Development Boards" in "Espressif").

Prerequisites:

- ESP-IDF development environment (ESP-IDF Setup) is set up.
- ESP32-H2-DevKitM-1 (https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32h2/esp32-h2-devkitm-1/index.html) is available.

Build and Run

This section demonstrates how to build and run the OpenThread CLI example on the **ESP32-H2 devkit**.

Step 1: Prepare the Environment

```
get_idf
cd $IDF_PATH/examples/openthread/ot_cli
```

Step 2: Set target to ESP32-H2

```
idf.py set-target esp32h2
```

Step 3 (Optional): Configure the Device

The example can run with the default configuration. `idf.py menuconfig` can be used for customized settings:

- Enabling Joiner Role: `Component Config → OpenThread → Thread Core Features → Enable Joiner`

Step 4: Connect to the Computer

Connect the ESP32-H2-DevKitM-1 to the computer using a USB cable.

Step 5: Build and Flash

```
idf.py build
idf.py -p <PORT_TO_ESP32_H2> flash monitor
```

Usage

Joining Thread Network

A Thread device can join the network using either a minimal dataset (providing only the Network Key) or a complete Active Operational Dataset.

Example 1: Joining with Minimal Provisioning (Network Key Only)

The device is initially provisioned with only the Network Key. After joining, the device synchronizes with the network and automatically receives the complete Active Operational Dataset.

```
dataset networkkey 00112233445566778899aabbccddeeff
dataset commit active
ifconfig up
thread start
```

Display the full Active Operational Dataset:

```
dataset active
```

Example 2: Joining with the Full Active Operational Dataset

In this method, all network parameters (such as PAN ID, channel, Mesh-Local Prefix, etc.) are provisioned upfront. The device is configured with the complete dataset before starting the Thread interface.

```
dataset set active
0e080000000000010000000300001835060004001fffe00208fe7bb701f5f1125d0708f
d75cbde7c6647bd0510b3914792d44f45b6c7d76eb9306eec94030f4f70656e54687265
61642d353833320102583204 10e35c581af5029b054fc904a24c2b27700c0402a0fff8
```

```
ifconfig up
thread start
```

Commissioning

Joiner device:

- Get the factory-assigned IEEE EUI-64 address (e.g., `744dbdfffe63f5c8`).

  ```
  eui64
  ```

- Bring the IPv6 interface up, enable the Thread Joiner role, and start the Thread protocol:

  ```
  ifconfig up
  joiner start J00MMM
  thread start
  ```

Commissioner device:

- Start the Commissioner role:

  ```
  commissioner start
  ```

- Add a joiner entry:

  ```
  commissioner joiner add <eui64|discerner> <pksd>
  ```

  Example:

  ```
  commissioner joiner add 744dbdfffe63f5c8 J00MMM or commissioner
  joiner add * J00MMM
  ```

- Display all Joiner entries in table format:

```
commissioner joiner table
```

The Joiner device receives the Network Key when joining the network.

Testing

The command `router table` prints a list of routers in a table format. Each router is identified by its Extended MAC.

The command `extaddr` returns the Extended MAC address of a device.

The `joiner id` command returns the Joiner ID used for commissioning.

Sending UDP packets

Running UDP Server:

```
udpsockserver open
udpsockserver bind 12345
udpsockserver status
```

Sending a UDP packet and closing the server:

```
udpsockserver send fdf9:2548:ce39:efbb:9612:c4a0:477b:349a 12346 hello
udpsockserver close
```

Running UDP Client:

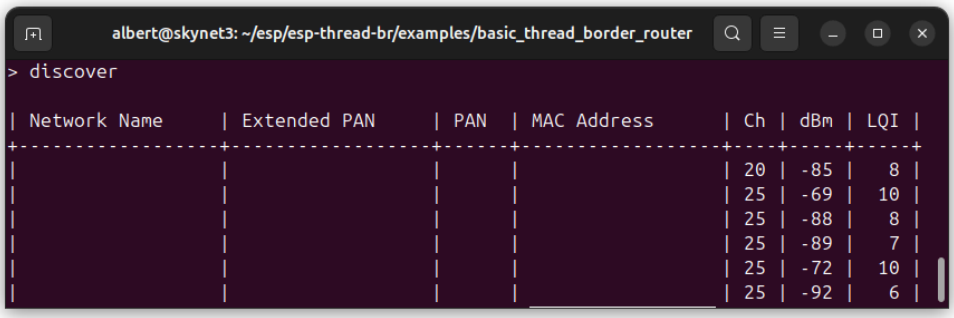
```
udpsockclient open # or udpsockclient open <port>, e.g., udpsockclient
open 12345
udpsockclient status
```

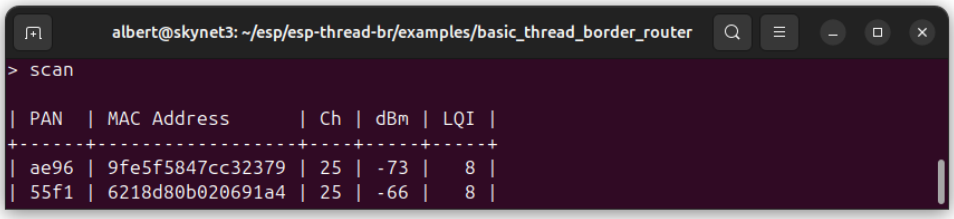
Sending a UDP packet and closing the client:

```
udpsockclient send fdf9:2548:ce39:efbb:79b9:4ac4:f686:8fc9 12346 hello
udpsockclient close
```

Scan and Discover

The `scan` command performs IEEE 802.15.4 scan to find nearby devices. The `discover` command performs an MLE Discovery operation to find Thread networks nearby.





References

- OpenThread CLI Command Reference (https://openthread.io/reference/cli/commands)
- Forming a Thread network on the Thread Border Router (https://openthread.io/codelabs/esp-openthread-hardware#3)
- How to set up a Command Line Interface on a thread device (https://mattercoder.com/codelabs/how-to-install-border-router-on-esp32/?index=..%2F..index#5)
- OpenThread CLI - Commissioning (https://github.com/openthread/ot-commissioner/tree/main/src/app/cli)
- OpenThread CLI - Operational Datasets (https://github.com/openthread/openthread/blob/main/src/cli/README_DATASET.md)
- ESP OT-CLI Example (https://github.com/espressif/esp-idf/tree/master/examples/openthread/ot_cli)

- Build and Run CLI device (https://docs.espressif.com/projects/esp-thread-br/en/latest/dev-guide/build_and_run.html#build-and-run-the-thread-cli-device)

CHIP-Tool

Overview

CHIP Tool (chip-tool) is a Matter controller implementation that allows to commission a Matter device into a network and to communicate with it. It was used for testing accessory devices.

CHIP Tool was built and installed on a **Raspberry Pi 4 Model B** (4GB RAM and 32GB SD card) running *Ubuntu Server for Raspberry Pi 24.04.1 LTS*. *Raspberry Pi Imager v1.8.5* was used to flash the Ubuntu image onto the SD card.

Matter v1.4.0.0 (https://github.com/project-chip/connectedhomeip/releases/tag/v1.4.0.0) SDK was used to build the CHIP-Tool application.

Raspberry Pi Setup

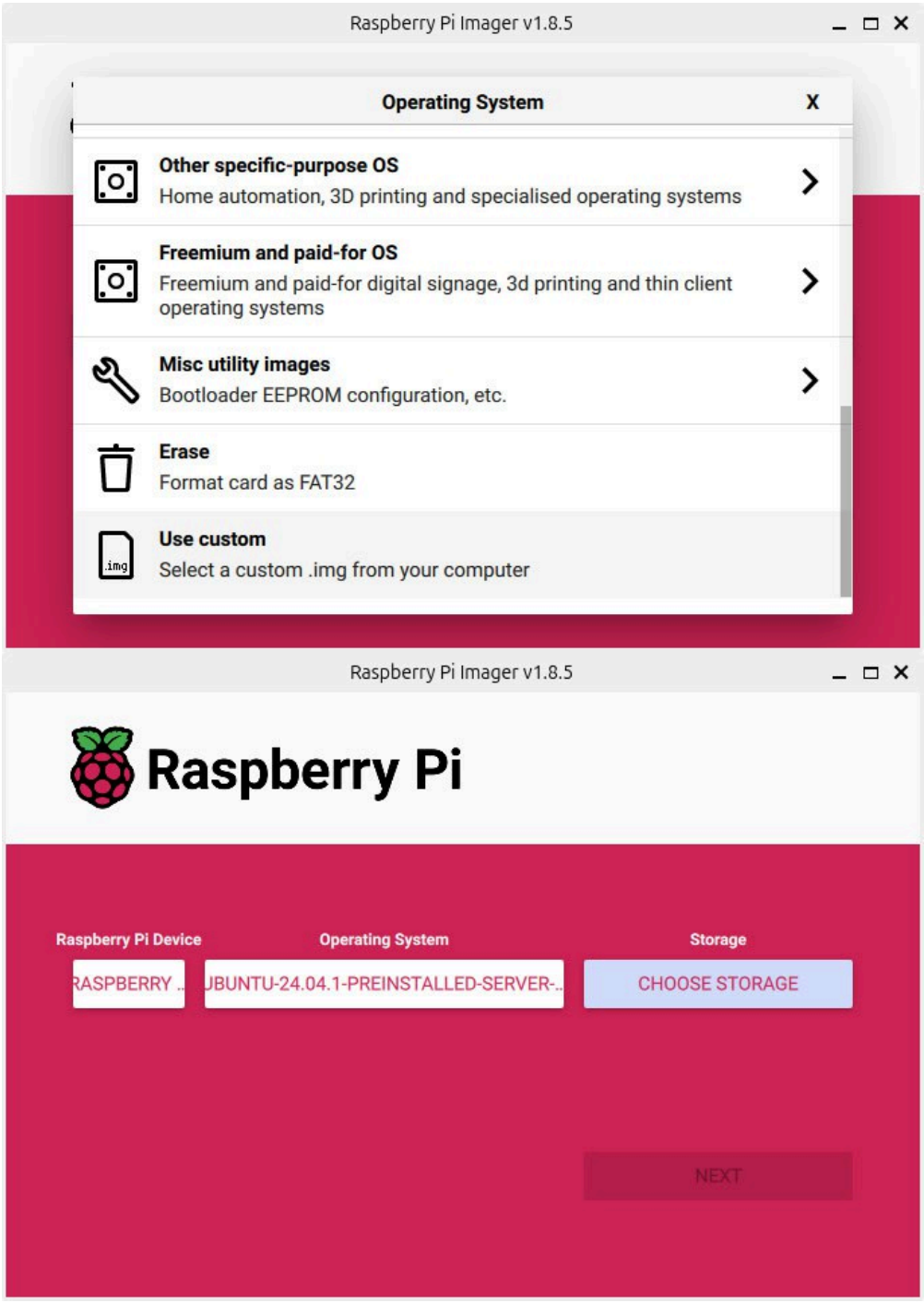
This section outlines the steps to set up a Raspberry Pi as a Matter controller using CHIP-Tool. The process is time-consuming.

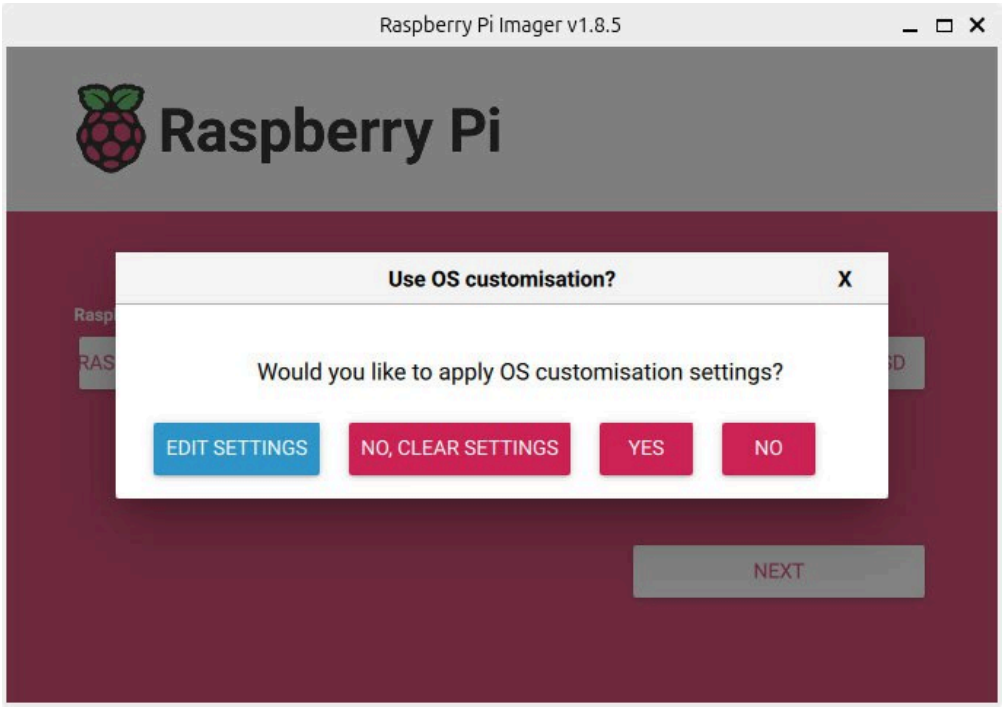
Step 1. Ubuntu installation

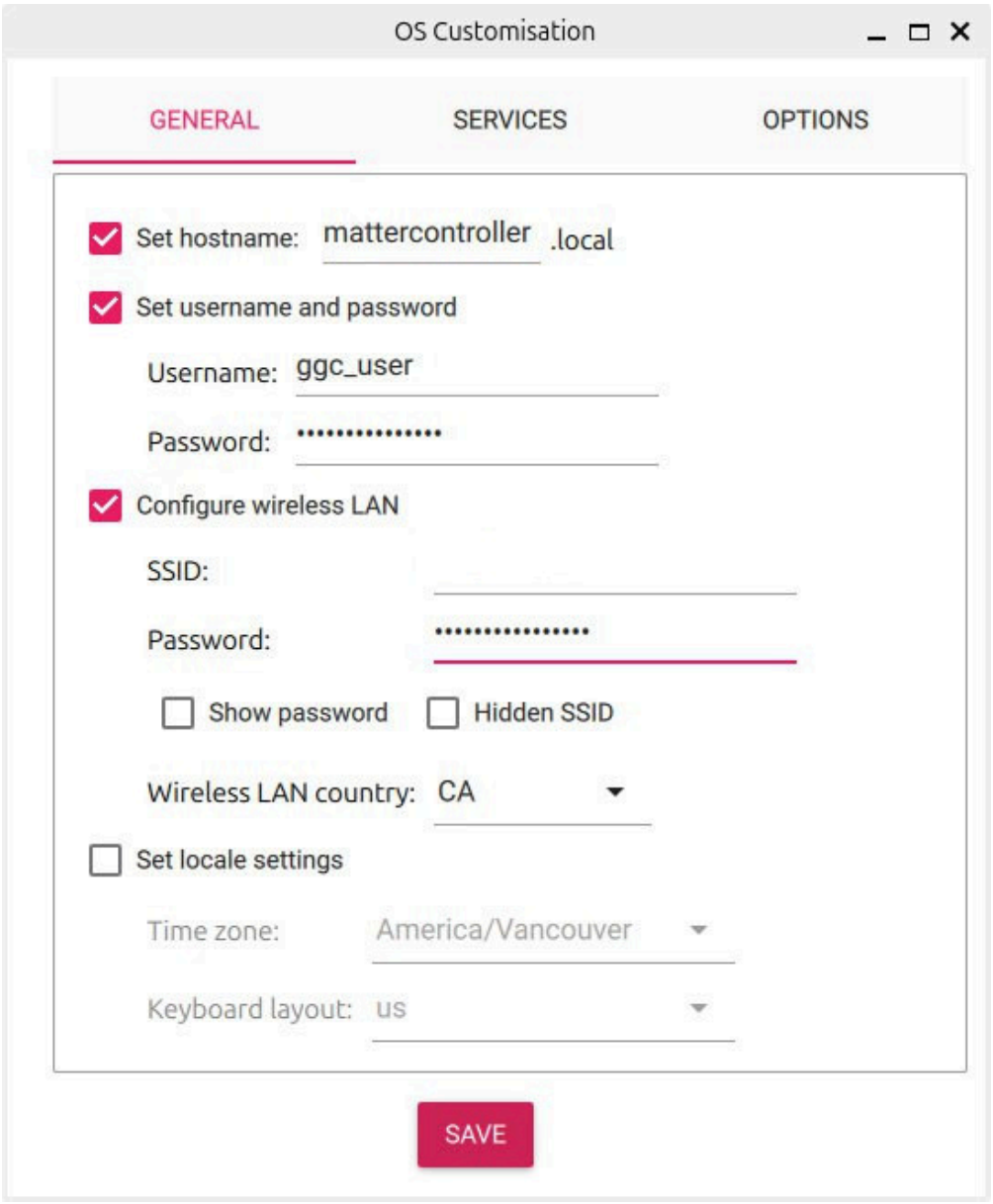
1. Install the OS
    1. Download the Ubuntu Server image from the Ubuntu website (https://ubuntu.com/download/raspberry-pi).
    2. Download and install Raspberry Pi Imager on Ubuntu Desktop:

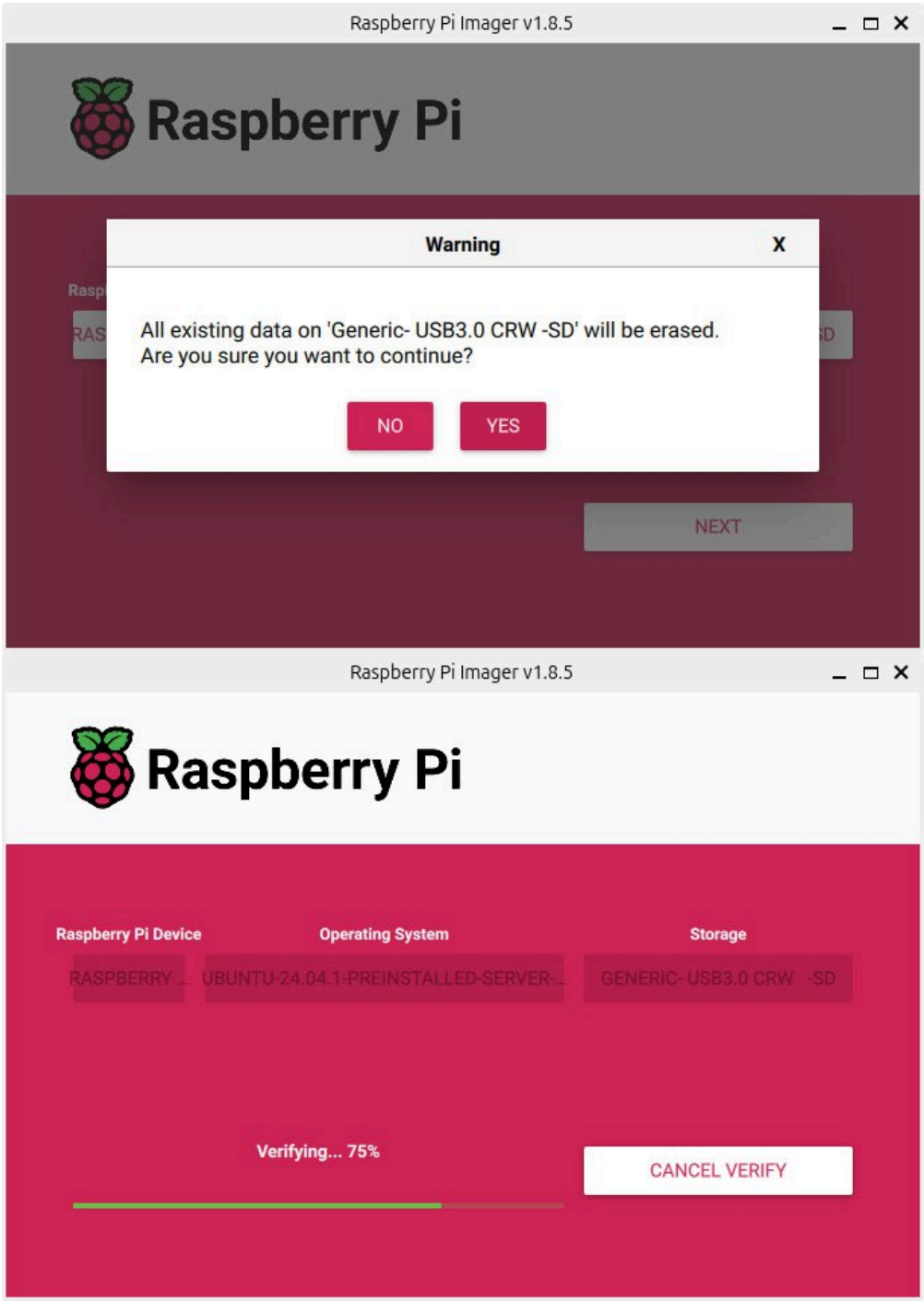
        ```
        sudo apt update
        sudo apt install rpi-imager
        ```

    3. Insert the SD card into card reader and connect it to the computer.
    4. Select the Ubuntu image and the SD card in Raspberry Pi Imager.
    5. Edit Settings (Wi-Fi and SSH credentials and enable SSH) and flash the image onto the SD card.


Raspberry Pi Imager v1.8.5
Operating System
X
Other specific-purpose OS
Home automation, 3D printing and specialised operating systems
Freemium and paid-for OS
Freemium and paid-for digital signage, 3d printing and thin client operating systems
Misc utility images
Bootloader EEPROM configuration, etc.
Erase
Format card as FAT32
Use custom
Select a custom .img from your computer
img
Raspberry Pi Imager v1.8.5
Raspberry Pi
Raspberry Pi Device
Operating System
Storage
RASPBERRY ..
UBUNTU-24.04.1-PREINSTALLED-SERVER-..
CHOOSE STORAGE
NEXT


Raspberry Pi Imager v1.8.5
Raspberry Pi
Use OS customisation?
X
Would you like to apply OS customisation settings?
EDIT SETTINGS
NO, CLEAR SETTINGS
YES
NO
NEXT

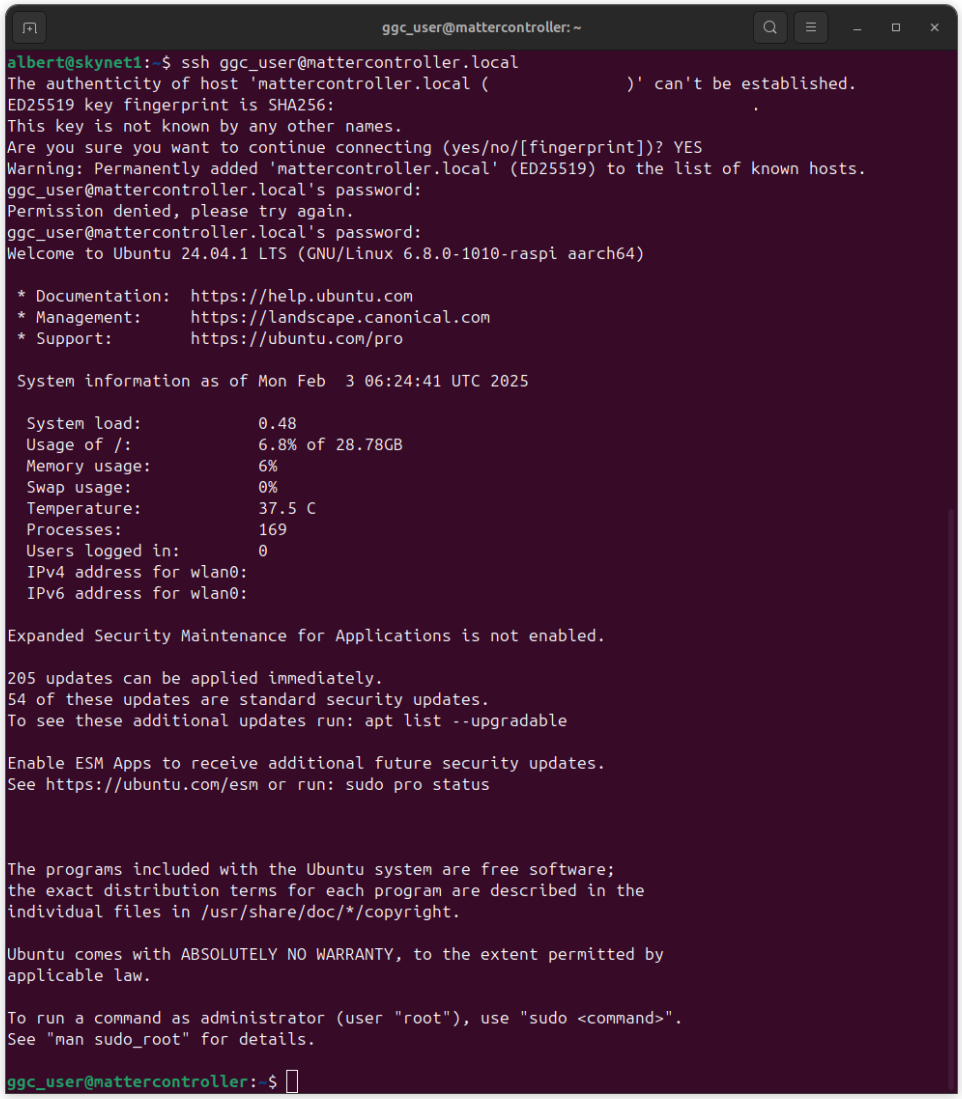

OS Customisation
GENERAL
SERVICES
OPTIONS
Set hostname:
mattercontroller
.local
Set username and password
Username:
ggc_user
Password:
Configure wireless LAN
SSID:
Password:
Show password
Hidden SSID
Wireless LAN country:
CA
Set locale settings
Time zone:
America/Vancouver
Keyboard layout:
us
SAVE



Step 2. Connecting to Raspberry Pi

Insert the SD card into the Raspberry Pi and power it on. It will automatically connect to Wi-Fi using the credentials provided during the installation process. It can take a few minutes for the Raspberry Pi to boot up. Once it is ready, connect to it using SSH with the credentials provided during the installation process.

```
ssh ggc_user@mattercontroller.local
```



Step 3. Installing Pre-requisites

1. Update the system and install Raspberry Pi-specific dependencies (https://github.com/project-chip/connectedhomeip/blob/master/docs/guides/BUILDING.md#installing-prerequisites-on-raspberry-pi-4):

```
sudo apt update
sudo apt upgrade
sudo apt-get install git gcc g++ pkg-config libssl-dev libdbus-1-dev
```

```
\
     libglib2.0-dev libavahi-client-dev ninja-build python3-venv
python3-dev \
     python3-pip unzip libgirepository1.0-dev libcairo2-dev
libreadline-dev \
     default-jre
sudo apt-get install bluez avahi-utils
sudo reboot
ssh ggc_user@mattercontroller.local
```

2. Edit the `bluetooth.service` unit:

```
sudo systemctl edit bluetooth.service
```

   Add the following content:

```
[Service]
ExecStart=
ExecStart=/usr/sbin/bluetoothd -E -P battery
Restart=always
RestartSec=3
```

3. Enable and Start the Bluetooth service:

```
sudo systemctl start bluetooth.service
```

4. Edit the dbus-fi.w1.wpa_supplicant1.service file:

```
sudo nano /etc/systemd/system/dbus-fi.w1.wpa_supplicant1.service
```

   Replace the line #11 with:

```
ExecStart=/usr/sbin/wpa_supplicant -u -s -O /run/wpa_supplicant
```

5. Edit the wpa_supplicant.conf file:

```
sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```

Add the following content:

```
ctrl_interface=DIR=/run/wpa_supplicant
update_config=1
```

6. Reboot the Raspberry Pi:

```
sudo reboot
ssh ggc_user@mattercontroller.local
```

Step 4. Building CHIP Tool

The application will require installation of Matter SDK on Raspberry Pi:

1. Clone the Matter SDK repository:

```
mkdir matter
cd ~/matter
git clone https://github.com/project-chip/connectedhomeip.git
cd connectedhomeip/
git checkout tags/v1.4.0.0
```

2. Prepare Matter SDK:

```
./scripts/checkout_submodules.py --shallow --platform  linux
source ~/matter/connectedhomeip/scripts/activate.sh
```

3. Build CHIP Tool using build_examples.py:

   The `targets` command retrieves supported build targets listed below:

   ```
   ./scripts/build/build_examples.py targets
   ```

   - **Ameba:**
     - `amebad-{all-clusters, all-clusters-minimal, light, light-switch, pigweed}`

- **ASR:** `asr-{asr582x, asr595x, asr550x}-{all-clusters, all-clusters-minimal, lighting, light-switch, lock, bridge, temperature-measurement, thermostat, ota-requestor, dishwasher, refrigerator}`
  - Optional flags: `[-ota] [-shell] [-no_logging] [-factory] [-rotating_id] [-rio]`
- **Android:** `android-{arm, arm64, x86, x64, androidstudio-arm, androidstudio-arm64, androidstudio-x86, androidstudio-x64}-{chip-tool, chip-test, tv-server, tv-casting-app, java-matter-controller, kotlin-matter-controller, virtual-device-app}`
  - Optional flags: `[-no-debug]`
- **Bouffalolab:** `bouffalolab-{bl602dk, bl704ldk, bl706dk, bl602-night-light, bl706-night-light, bl602-iot-matter-v1, xt-zb6-devkit}-light`
  - Optional flags: `[-ethernet] [-wifi] [-thread] [-easyflash] [-littlefs] [-shell] [-mfd] [-rotating_device_id] [-rpc] [-cdc] [-mot] [-resetcnt] [-memmonitor] [-115200] [-fp]`
- **Texas Instruments (TI):**
  - `cc32xx-{lock, air-purifier}`
  - `ti-cc13x4_26x4-{lighting, lock, pump, pump-controller}`
  - Optional flags: `[-mtd] [-ftd]`
- **Cypress (Infineon):** `cyw30739-{cyw30739b2_p5_evk_01, cyw30739b2_p5_evk_02, cyw30739b2_p5_evk_03, cyw930739m2evb_01, cyw930739m2evb_02}-{light, light-switch, lock, thermostat}`
- **Silicon Labs (EFR32):** `efr32-{brd2704b, brd4316a, brd4317a, brd4318a, brd4319a, brd4186a, brd4187a, brd2601b, brd4187c, brd4186c, brd2703a, brd4338a, brd2605a}-{window-covering, switch, unit-test, light, lock, thermostat, pump, air-quality-sensor-app}`
  - Optional flags: `[-rpc] [-with-ota-requestor] [-icd] [-low-power] [-shell] [-no-logging] [-openthread-mtd] [-heap-monitoring] [-no-openthread-cli] [-show-qr-code] [-wifi] [-rs9116] [-wf200] [-siwx917] [-ipv4] [-additional-data-advertising] [-use-ot-lib] [-use-ot-coap-lib] [-no-version] [-skip-rps-generation]`

- **Espressif (ESP32):** `esp32-{m5stack, c3devkit, devkitc, qemu}-{all-clusters, all-clusters-minimal, energy-management, ota-provider, ota-requestor, shell, light, lock, bridge, temperature-measurement, tests}`
  - Optional flags: `[-rpc] [-ipv6only] [-tracing] [-data-model-disabled] [-data-model-enabled]`
- **General:**
  - `genio-lighting-app`
- **Linux:**
  - `linux-fake-tests`
  - Optional flags: `[-mbedtls] [-boringssl] [-asan] [-tsan] [-ubsan] [-libfuzzer] [-ossfuzz] [-pw-fuzztest] [-coverage] [-dmalloc] [-clang] linux-arm64-{rpc-console, all-clusters, all-clusters-minimal, chip-tool, thermostat, java-matter-controller, kotlin-matter-controller, minmdns, light, light-data-model-no-unique-id, lock, shell, ota-provider, ota-requestor, simulated-app1, simulated-app2, python-bindings, tv-app, tv-casting-app, bridge, fabric-admin, fabric-bridge, tests, chip-cert, address-resolve-tool, contact-sensor, dishwasher, microwave-oven, refrigerator, rvc, air-purifier, lit-icd, air-quality-sensor, network-manager, energy-management}`
  - Optional flags: `[-nodeps] [-nlfaultinject] [-platform-mdns] [-minmdns-verbose] [-libnl] [-same-event-loop] [-no-interactive] [-ipv6only] [-no-ble] [-no-wifi] [-no-thread] [-no-shell] [-mbedtls] [-boringssl] [-asan] [-tsan] [-ubsan] [-libfuzzer] [-ossfuzz] [-pw-fuzztest] [-coverage] [-dmalloc] [-clang] [-test] [-rpc] [-with-ui] [-evse-test-event] [-enable-dnssd-tests] [-disable-dnssd-tests] [-chip-casting-simplified] [-data-model-check] [-data-model-disabled] [-data-model-enabled] [-check-failure-die]`
  - `linux-arm64-efr32-test-runner`
  - Optional flags: `[-clang]`

The `linux-arm64-chip-tool` and `linux-arm64-all-clusters` targets were selected from the Linux section, with the `-ipv6only` and `-platform-mdns` flags:

```
./scripts/build/build_examples.py --target linux-arm64-chip-tool-
ipv6only-platform-mdns gen
cd ~/matter/connectedhomeip/out/linux-arm64-chip-tool-ipv6only-
platform-mdns
ninja -j 1 & disown
```

Building the CHIP-Tool is a time-consuming process. To prevent SSH session timeouts, ninja was executed in the background. The job count was limited to 1 (`-j 1`) to reduce memory usage. `tail -f nohup.out` can be used to monitor the output.

4. Move the built CHIP Tool:

```
mv ~/matter/connectedhomeip/out/linux-arm64-chip-tool-ipv6only-
platform-mdns/chip-tool ~/matter/chip-tool
rm -rf ~/matter/connectedhomeip/out/linux-arm64-chip-tool-ipv6only-
platform-mdns
```

Step 5. Testing

Running CHIP Tool:

```
cd ~/matter
./chip-tool
```

Commissioning

This section describes the Matter commissioning (Matter Commissioning) using CHIP Tool.

The following command should be run on the Commissionee to print the static configuration that includes the **PIN code** and **Discriminator** (*0xf00* is *3840* in decimal):

```
matter config
```

Commissioning to Wi-Fi over BLE

A Commissionee joins an existing IP network over Bluetooth LE before being commissioned into a Matter network.

The following CHIP-Tool command starts commissioning onto a **Wi-Fi** network over BLE:

```
./chip-tool pairing ble-wifi 0x1122 SSID WIFIPASS 20202021 3840
./chip-tool pairing ble-wifi <NODE_ID_TO_ASSIGN> <SSID> <PASSWORD>
<PIN> <DISCRIMINATOR>
```

The parameters are defined as follows:

- `<NODE_ID_TO_ASSIGN>` is the node ID assigned to the device being commissioned, which can be a decimal or hexadecimal number prefixed with '0x'.
- `<SSID>` is the Wi-Fi SSID, provided as a plain string or in hexadecimal format as 'hex:XXXXXXXX', where each byte of the SSID is represented as two-digit hexadecimal numbers.
- `<PASSWORD>` is the Wi-Fi password, given as a plain string or hex data.
- `<PIN>` is the PIN code for authentication.
- `<DISCRIMINATOR>` is the discriminator value.

CHIP-Tool starts the commissioning process, which includes the following steps:

1. **Initialization**
   - Initializes storage and loads key-value store (KVS) configurations.
   - Detects network interfaces and identifies the WiFi interface.
   - Sets up UDP transport manager and BLE transport layers.
2. **Fabric and Node Setup**
   - Loads the fabric table, which contains information about existing secure networks.
   - Creates a new fabric with a unique Fabric ID and Node ID.
3. **BLE Scanning and Connection**
   - Scans for nearby BLE devices.
   - Identifies the correct device using a discriminator match.

- Establishes a BLE connection with the device.

4. **Secure Session Establishment**
    - Performs a PASE (Password Authenticated Session Establishment) handshake over BLE.
    - Exchanges secure messages to establish an encrypted session.
    - Marks the session as "Active" upon success.
5. **Commissioning Process**
    - Reads the device's attributes and capabilities.
    - Arms a fail-safe mechanism to prevent accidental changes.
    - Configures the device based on regional regulatory requirements.
6. **Certificate Exchange**
    - The device provides PAI (Product Attestation Intermediate) and DAC (Device Attestation Certificate).
    - The system verifies the certificates and device authenticity.
7. **Attestation and Verification**
    - Validates the device's attestation data to ensure a trusted identity.
    - Performs a revocation check to confirm the device's legitimacy.
8. **Operational Certificate Signing**
    - The device generates a CSR (Certificate Signing Request).
    - The system issues a NOC (Node Operational Certificate) to authenticate the device in the network.
9. **Finalizing the Pairing**
    - Sends the root certificate to the device.
    - Marks the pairing process as "Success," making the device a trusted member of the

Matter network.

Commissioning to Thread over BLE

The following CHIP-Tool command initiates commissioning onto a **Thread** network over BLE:

```
./chip-tool pairing ble-thread 0x1122 20202021 3840
./chip-tool pairing ble-thread <NODE_ID_TO_ASSIGN> hex:
<OPERATIONAL_DATASET> <PIN> <DISCRIMINATOR>
```

The parameters are defined as follows:

- `<NODE_ID_TO_ASSIGN>` is the node ID assigned to the device being commissioned, which can be a decimal number or a hexadecimal number prefixed with '0x'.
- `<OPERATIONAL_DATASET>` is the Thread Operational Dataset (Thread), which contains the network credentials needed to join a Thread network. It is provided in hexadecimal format (`hex:XXXXXXXX`), where each byte of the dataset is represented as two-digit hexadecimal numbers.
- `<PIN>` is the PIN code for authentication.
- `<DISCRIMINATOR>` is the discriminator value.

Commissioning over Existing IP Network

A Commissionee already connected to a Wi-Fi, Ethernet, or Thread network can be commissioned without BLE discovery, using **mDNS service discovery** instead.

The following CHIP-Tool command commissions a device that is already connected to a **Wi-Fi** or **Ethernet** network:

```
./chip-tool pairing onnetwork 0x1122 20202021
./chip-tool pairing onnetwork <NODE_ID_TO_ASSIGN> <PIN>
```

The parameters are defined as follows:

- `<NODE_ID_TO_ASSIGN>` is the **node ID** assigned to the device being commissioned.

- `<PIN>` is the PIN code for authentication.

The following CHIP-Tool command commissions a device that is already connected to a **Thread** network:

```
./chip-tool pairing onnetwork-thread 0x1122 hex:<OPERATIONAL_DATASET>
20202021
./chip-tool pairing onnetwork-thread <NODE_ID_TO_ASSIGN> hex:
<OPERATIONAL_DATASET> <PIN>
```

The parameters are defined as follows:

- `<NODE_ID_TO_ASSIGN>` is the **node ID** assigned to the device being commissioned.
- `<OPERATIONAL_DATASET>` is the **Thread Operational Dataset**, which contains the network credentials for joining a Thread network.
- `<PIN>` is the **PIN code** for authentication.

Removing a Device from Fabric

The following CHIP-Tool command removes a device from the Matter Fabric:

```
./chip-tool pairing unpair 0x1122
./chip-tool pairing unpair <NODE_ID_TO_ASSIGN>
```

The parameters are defined as follows:

- `<NODE_ID_TO_ASSIGN>` is the **node ID** assigned to the device being removed.

Reading Device Attributes

The following command is used to read the current state of the "on/off" attribute from the target device:

```
./chip-tool onoff read on-off 0x1122 1
```

In this command:

- `onoff` is the cluster for controlling the power state.
- `read` is a request to retrieve the current state.
- `on-off` is the specific attribute being queried.
- `0x1122` is the hexadecimal node ID of the device.
- `1` is the endpoint number on the target device.

Similarly, the command below reads the measurement value from the temperature sensor:

```
./chip-tool temperaturemeasurement read measured-value 0x1122 1
```

Sending Commands to Device

The following command is used to turn on and toggle the device:

```
./chip-tool onoff on 0x1122 1
./chip-tool onoff toggle 0x1122 1
```

Subscribing to Attributes and Events

This section describes how to subscribe ("Subscribing to events or attributes" in "Matter Controllers") to attributes and events in a Matter device using CHIP-Tool.

Subscribing to Attributes

Display all the attributes available for subscription in a given cluster:

```
./chip-tool <cluster-name> subscribe
```

Subscribe to an attribute:

```
./chip-tool <cluster-name> subscribe <argument> <min-interval> <max-interval> <node_id> <endpoint_id>
```

In this command:

- **cluster-name** is the name of the cluster.
- **argument** is the name of the chosen argument.
- **min-interval** specifies the minimum number of seconds that must elapse since the last report for the server to send a new report.
- **max-interval** specifies the number of seconds that must elapse since the last report for the server to send a new report.
- **node-id** is the user-defined ID of the commissioned node.
- **endpoint_id** is the ID of the endpoint where the chosen cluster is implemented.

For example:

```
./chip-tool doorlock subscribe lock-state 5 10 1 1
./chip-tool temperaturemeasurement subscribe measured-value 3 10 1 1
./chip-tool relativehumiditymeasurement subscribe measured-value 3 10 1
2
./chip-tool occupancysensing subscribe occupancy 3 10 1 3
```

Subscribing to Events

Display all the events available for subscription in a given cluster:

```
./chip-tool <cluster-name> subscribe-event
```

Subscribe to an event:

```
./chip-tool <cluster-name> subscribe-event <event-name> <min-interval>
<max-interval> <node_id> <endpoint_id>
```

In this command:

- **cluster-name** is the name of the cluster.
- **event-name** is the name of the chosen event.

- **min-interval** specifies the minimum number of seconds that must elapse since the last report for the server to send a new report.
- **max-interval** specifies the number of seconds that must elapse since the last report for the server to send a new report.
- **node_id** is the user-defined ID of the commissioned node.
- **endpoint_id** is the ID of the endpoint where the chosen cluster is implemented.

For example:

```
./chip-tool doorlock subscribe-event door-lock-alarm 5 10 1 1
```

References

- How to Install Matter on RPi (https://mattercoder.com/codelabs/how-to-install-matter-on-rpi/)
- Setting up the Matter Hub (Raspberry Pi) (https://docs.silabs.com/matter/latest/matter-thread/raspi-img)
- CHIP-Tool - Commissioning Device over BLE (https://github.com/project-chip/connectedhomeip/blob/master/examples/chip-tool/README.md#commission-a-device-over-ble)
- CHIP-tool Source Code (https://github.com/project-chip/connectedhomeip/tree/master/examples/chip-tool)
- Silicone Labs' Matter Commissioning Guide (https://docs.silabs.com/matter/2.2.1/matter-overview-guides/matter-commissioning)

Orchestrator

Overview

Orchestrators drive system automation by establishing network infrastructure, integrating new devices, and continuously optimizing environmental conditions for plant growth.

Orchestrators combine the key features of OpenThread CLI ("OpenThread CLI" in "Thread") and Matter Controller:

- Create and manage Thread (Thread) networks and commission (Thread Commissioning) devices to join them.
- Establish Matter (Matter) Fabrics and commission (Matter Commissioning) devices to join them.
- Monitor sensor data from the Grow Chamber and adjust actuators in real time.

Hardware

The ESP32-C6 ("Development Boards" in "Espressif") supports both Thread and Wi-Fi protocols but cannot manage both communications at the same time. This limitation makes it unsuitable for use as a Thread Border Router. However, its dual-protocol support makes it ideal for functioning as an Orchestrator Device.

Firmware

The firmware for the Orchestrator was developed using the ESP-IDF framework ("ESP-IDF Framework" in "Espressif").

Project Bootstrap

Follow the following steps from the ESP32 Project Workflow (ESP32 Project Workflow) guide:

- Step 1: Set Up ESP-IDF Environment ("Step 1: Set Up ESP-IDF Environment" in "ESP32 Project Workflow")

- Step 2: Create a New Project ("Step 2: Create a New Project" in "ESP32 Project Workflow")
- Step 3: Set ESP32 as the target device ("Step 3: Set ESP32 as the target device" in "ESP32 Project Workflow")
- Step 4: Open the Project in CLion IDE ("Step 4: Open the Project in CLion IDE" in "ESP32 Project Workflow")
- Step 5: Create Default Configuration ("Step 5: Create Default Configuration" in "ESP32 Project Workflow")
- Step 6: Update CMakeLists.txt ("Step 6: Update CMakeLists.txt" in "ESP32 Project Workflow")

Development of Components

The following sections describe the development process of the components implemented in the Orchestrator firmware:

- Thread Interface (Thread Interface)
- Matter Interface (Matter Interface)

General System Initialization

The main function initializes the essential components:

- The NVS (https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/nvs_flash.html) library stores key-value pairs in the device's flash memory.
- The Event Loop (https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/esp_event.html) allows components to declare events and register handlers, decoupling event producers from consumers. An event loop processes these events by invoking the appropriate handlers. The `default event loop` created by `esp_event_loop_create_default` is shared by multiple system components.

Components Initialization

- Thread Interface (Thread Interface)
- Matter Interface (Matter Interface)

Deployment

Follow the following steps from the ESP32 Project Workflow (ESP32 Project Workflow) guide:

- Step 7: Build the Project ("Step 7: Build the Project" in "ESP32 Project Workflow")
- Step 8: Determine Serial Port ("Step 8: Determine Serial Port" in "ESP32 Project Workflow")
- Step 9: Flash Project to Target ("Step 9: Flash Project to Target" in "ESP32 Project Workflow")
- Step 10: Launch IDF Monitor ("Step 10: Launch IDF Monitor" in "ESP32 Project Workflow")

Thread Interface

Overview

This section outlines the development of the **Thread Interface** component, which provides the features and functions to create Thread (Thread) networks and commission (Thread Commissioning) devices to join them.

This component is part of the Orchestrator (Orchestrator) firmware.

Development

Bootstrapping the Component

The command below creates a new component named `thread_interface` inside the `components` directory.

```
idf.py create-component thread_interface -C components
```

The `CONFIG_OPENTHREAD_ENABLE` configuration option has to be set to `y` in the `sdkconfig.defaults` file in order to enable the OpenThread component.

Component Initialization

The component encapsulates all OpenThread setup within a dedicated FreeRTOS task.

It initializes the essential components required for OpenThread operation:

- The Event VFS (https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/vfs.html#eventfd) provides a lightweight mechanism for representing events as file descriptors, similar to Linux's eventfd, which enables tasks to signal and wait for events. It is part of the VFS (https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/storage/vfs.html) Component that offers a unified interface for interacting with various file systems.
- The ESP-NETIF (https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_netif.html) library is an abstraction layer that provides thread-

safe APIs for managing network interfaces on top of a TCP/IP stack.

It initializes the OpenThread stack using default radio, host, and port configurations, creates an OpenThread network interface via `esp_netif`, and sets up additional features such as the CLI ("OpenThread CLI" in "Thread") and state indicator when enabled.

Once configured, it launches the OpenThread main loop and performs cleanup upon completion.

References

OpenThread SDK:

- How to Write an OpenThread Application (https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/openthread.html#how-to-write-an-openthread-application)

Android SDK

- Android Thread Commissioner (https://github.com/openthread/ot-commissioner/tree/main/android)
- Thread Network SDK for Android (https://developers.home.google.com/thread)

Matter Interface

Reading

```
#include "esp_matter_controller_read_command.h"
#include "esp_matter_client.h"

void response_callback(const esp_matter::client::response_t *response,
void *priv_data)
{
    // Process the response data
    if (response->status == ESP_OK) {
    // Handle successful response
    } else {
    // Handle error
    }
}

void client_request_callback(peer_device_t *peer_device,
request_handle_t *req_handle, void *priv_data)
{
    if (req_handle->type == READ_ATTR) {
    esp_matter::client::interaction::read::send_request(peer_device,
&req_handle->attribute_path, 1, NULL, 0, response_callback);
    }
}

esp_matter::client::set_request_callback(client_request_callback,
NULL);
```

References

- ESP-Matter Controller Example (https://github.com/espressif/esp-matter/tree/main/examples/controller)

- ESP-Matter Controller Docs (https://docs.espressif.com/projects/esp-matter/en/latest/esp32/developing.html#matter-controller)
- ESP-Matter Controller Component (https://github.com/espressif/esp-matter/tree/main/components/esp_matter_controller/commands)

Control Panel

Overview

A **Control Panel** is a GUI application that farmers use to configure and manage system components, as well as monitor and adjust environmental conditions.

The environment can be configured for specific crops. When setting up a new plant, farmers can choose from predefined parameters suited to that crop. As the plant grows, they can manually adjust settings.