ELECTRONICS
SOFTWARE
MECHATRONICS
MANUFACTURING

Technical & Functional analysis

Breakout game with accelerometer controller using Thread/Matter

Date : 6/7/2024

Contacts : Dekimo : Gysen Bart

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Table of Contents

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

1. Project overview

This project is an interactive breakout game controlled by motion with a wireless controller. The controller is connected to a Thread network and communicates with the Matter protocol. The breakout game is displayed on a tv using a SBC (Single Board Computer) that is also connected to a Thread network.

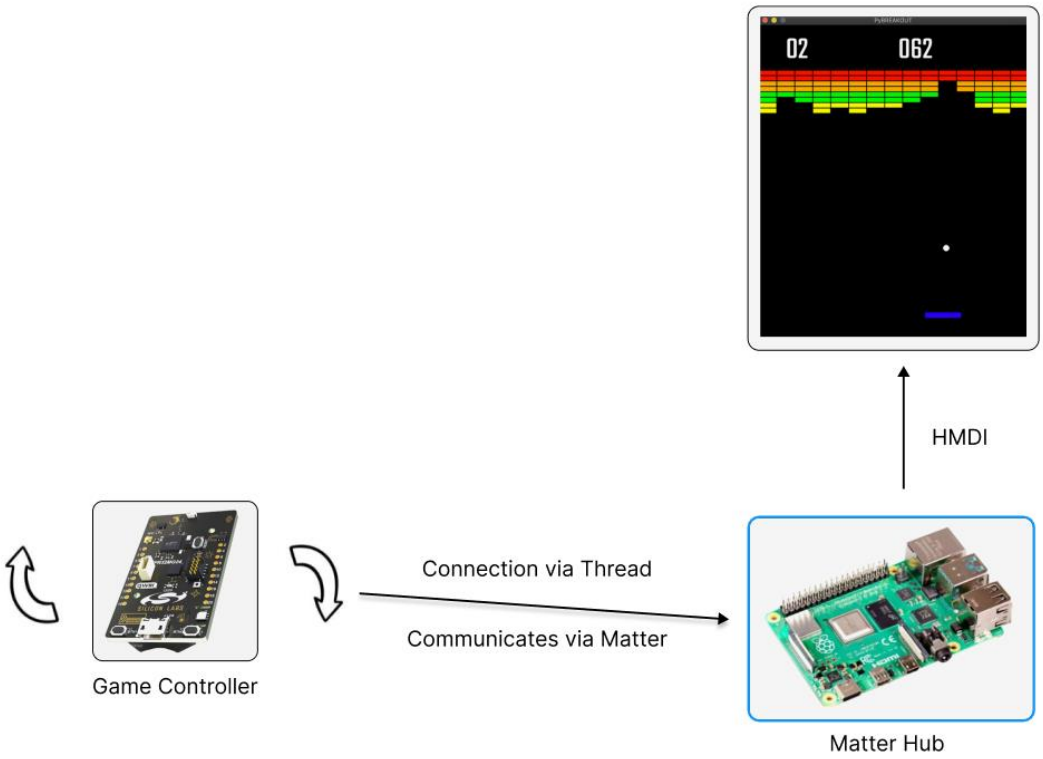


Figure 1 Project overview

1.1. Matter protocol

Matter, formerly known as “Project CHIP” (Connected Home over IP) is an open-source, royalty-free connectivity standard designed to make it easier for various smart devices to communicate with each other. It aims to create a unified standard for the Internet of Things devices. A Matter device can connect to a network with Thread, WIFI and Ethernet.

1.2. OpenThread network protocol

OpenThread is an low-power mesh networking protocol that is based on IPv6. OpenThread is an open standard and is built for IoT applications. It uses 6LoWPAN which uses IEEE 802.15.4 (2.4Ghz) wireless protocol with mesh communication.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

2. Game controller

The wireless controller must be equipped with:

- 2.4Ghz wireless controller used for the OpenThread connection
- Accelerometer used to interact with the game (moving the bar)
- Button used for interact with the game (pause, start, etc.)
- A battery to provide power to the controller

2.1. EFR32MG24 - BRD2601B Dev Kit

The EFR32MG24 Dev Kit board (BRD2601B) supports all the elements that is necessary for the controller. So there is no need to develop a PCB with all this components.

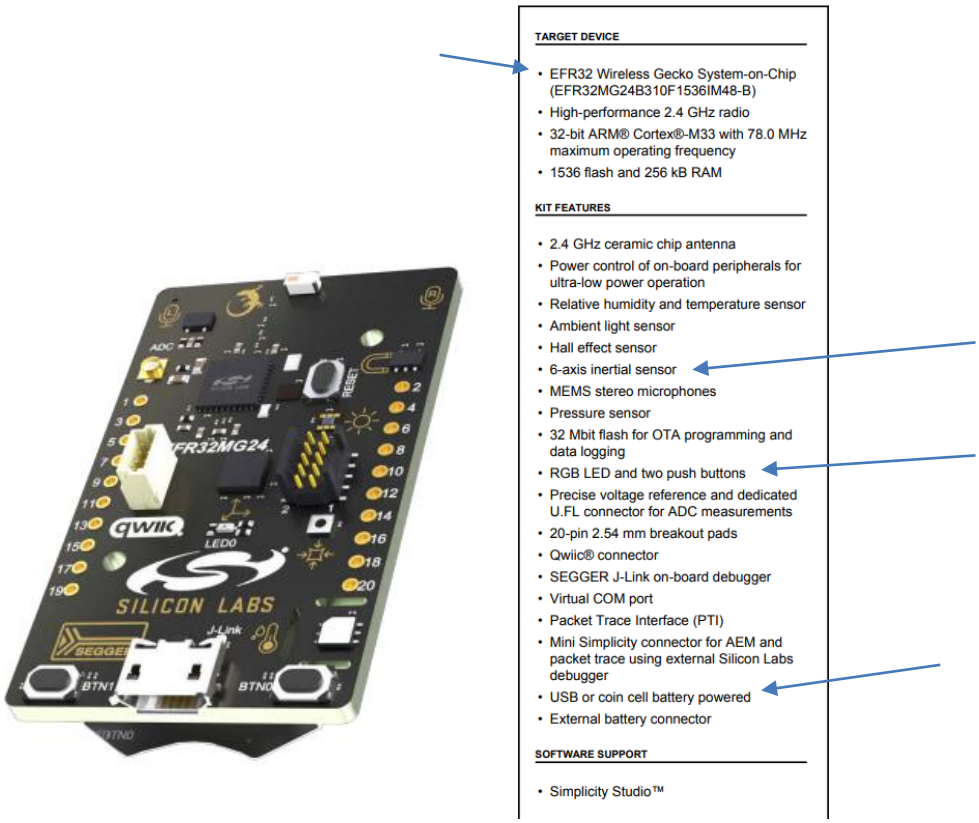


Figure 2 EFR32MG24

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

2.2. EFR32MG24 Wireless SoC

The development board uses the EFR32MG24 wireless SoC (System on a Chip). This SoC is ideal for meshing wireless solutions using Matter and OpenThread. It provides al the built in features that are relevant for this project, like:

- High performance 32-bit 78 MHz Arm Cortex®-M33 processor
- 1536 kB flash and 256 kB of RAM
- High performance 2.4GHz Radio
- OpenThread and Matter support
- Security features for protection against hardware and software attacks
- A wide range of peripherals like I²C, SPI, USART, ADC, Timers, GPIO's, etc.

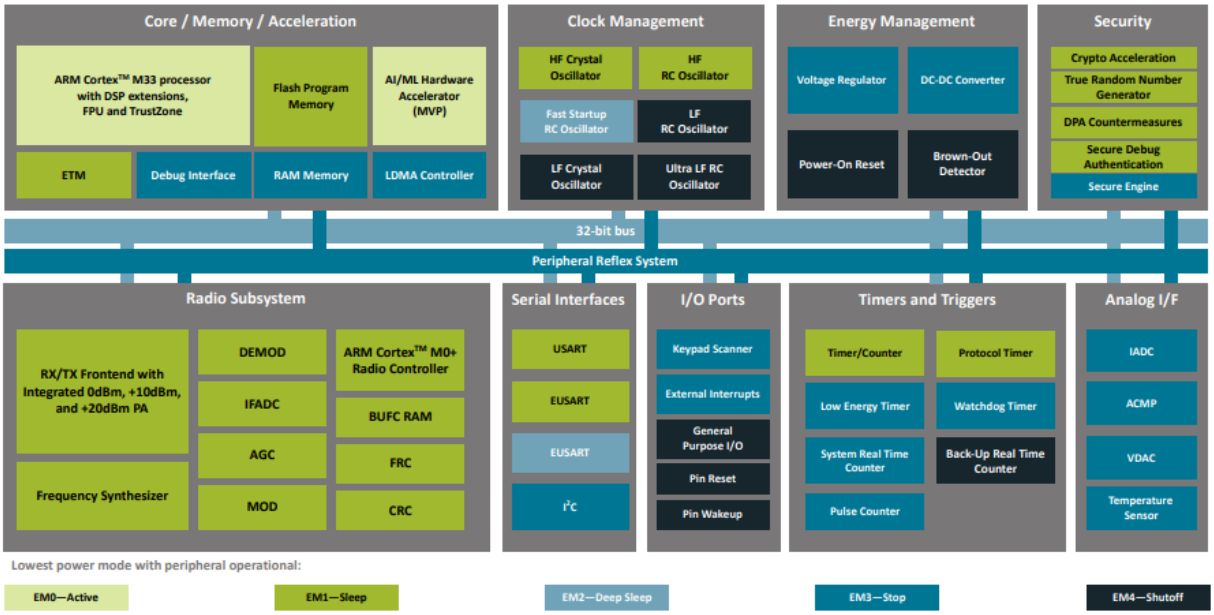


Figure 3 EFR32MG24 Block schematic

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

2.3. ICM-20689 6-Axis sensor

The development board contains a 6-axis sensor ICM-20689. This 6-axis sensor combines a 3-axis gyroscope and a 3-axis accelerometer. It detects acceleration and angular rate in the X, Y and Z axes.

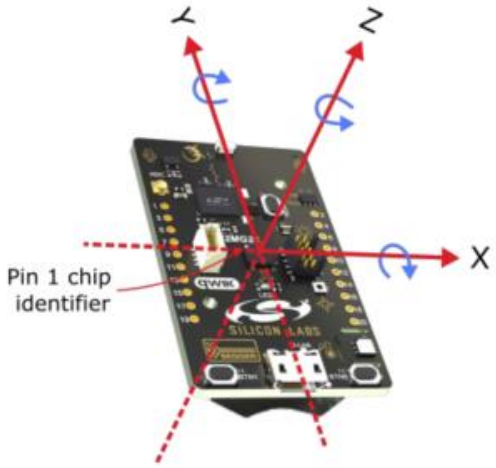


Figure 4 Built-in 6-Axis sensor

In the development board the sensor is connected and communicates over SPI. The SPI lines are interrupted through a switch to prevent power consumption when not used. Before the sensor can be used in the application it must be enabled by setting PC09 high.

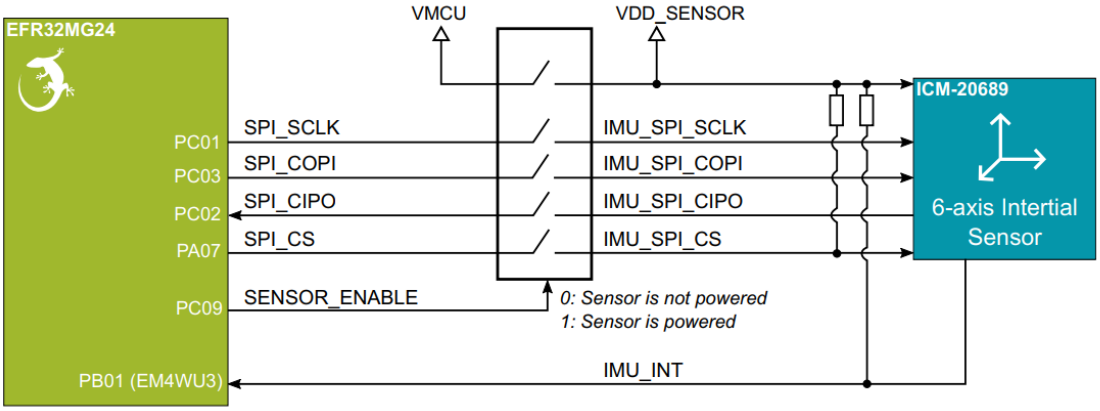


Figure 5 ICM-20689 SPI pins

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

2.4. User Button

The development board has two user buttons, BTN0 and BTN1. They are directly connected to the EFR32MG24 SoC and are debounced by RC filters.

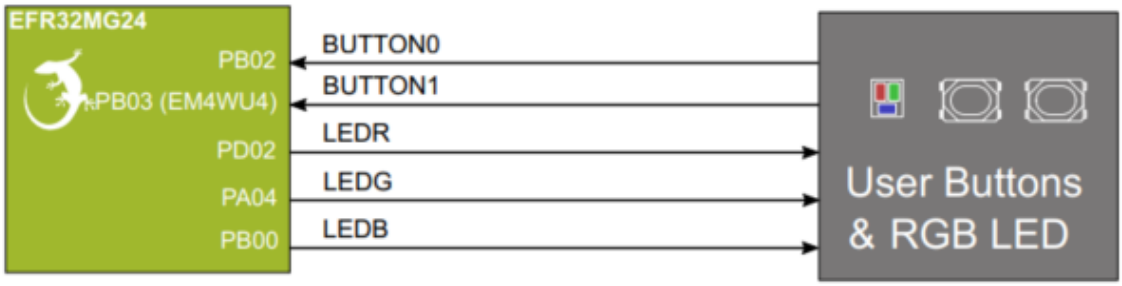


Figure 6 User buttons & RBG Led pins

The buttons are connected to the pins PB02 and PB03. The pins are also available on the expansion header. The buttons on the board can be used during the development and later on we can use buttons in a case that are connected to the expansion header.

Pin	Connection	EXP Header Function	Shared Feature
Right Side Breakout Pins			
2	VMCU	EFR32MG24 voltage domain, included in AEM measurements.	
4	PC03	SPI_COPI	IMU & Flash
6	PC02	SPI_CIPO	IMU & Flash
8	PC01	SPI_SCLK	IMU & Flash
10	PA07	SPI_CS	IMU_SPI_CS (when SENSOR_ENABLE = 1)
12	PA05	UART_TX	VCOM & Mini Simplicity
14	PA06	UART_RX	VCOM & Mini Simplicity
16	PC05	I2C_SDA	Qwiic I²C bus
18	5V	Board USB voltage	
20	3V3	Board controller supply	
Left Side Breakout Pins			
1	GND	Ground	
3	**PB02**	**GPIO**	**BTN0**
5	AIN3	ADC Negative Input	
7	AIN2	ADC Positive Input	
9	PB00	GPIO	LEDB
11	**PB03**	**GPIO**	**BTN1**
13	PD02	GPIO	LEDR
15	PC04	I2C_SCL	Qwiic I²C bus
17	BOARD_ID_SCL	Connected to Board Controller for identification of add-on boards.	
19	BOARD_ID_SDA	Connected to Board Controller for identification of add-on boards.	

Figure 7 User buttons pins description

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

3. (Open) Thread

Thread is a low-power IPv6 mesh networking standard for IoT devices. It is designed for low-power Internet of things devices. The low-power aspect is important for battery-powered smart home devices. However, it's also low-bandwidth, making it ideal for applications that don't send a lot of data, like switches or motion sensors.

Thread uses the same technology as Zigbee (IEEE 802.15.4) but provides IP connectivity similar to Wi-Fi. Unlike Zigbee, Thread by itself does not allow controlling devices: It is just a communication protocol. To control the Thread devices, a higher-level protocol is required e.g. Matter. Thread devices use the IPv6 standard to communicate both inside and outside the mesh network.

3.1. Thread vs OpenThread

The Thread protocol specification is available at no cost; however, this requires agreement and continued adherence to an End-User License Agreement (EULA), which states that "Membership in Thread Group is necessary to implement, practice, and ship Thread technology and Thread Group specifications.

OpenThread released by Google is an open-source implementation of Thread®. Google has released OpenThread to make the networking technology used in Google Nest products more broadly available to developers, in order to accelerate the development of products for the connected home and commercial buildings.

With a narrow platform abstraction layer and a small memory footprint, OpenThread (OT) is highly portable. It supports both System-on-Chip (SoC) and Co-Processor (RCP, NCP) designs.

3.2. RCP

A OTBR supports a Radio Co-Processor (RCP) design. In this design the core of OpenThread lives on the host processor with only a minimal MAC layer on the processor with the Thread radio. The host processor typically doesn't sleep in this design, in part to ensure reliability of the Thread network. The advantage of this design is that more resources are available on the more powerful host processor. Communication between the RCP and the host processor is managed by OpenThread Daemon over the Spinel protocol.

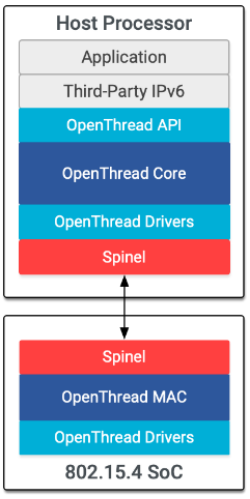


Figure 8 RCP

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

3.3. NCP

The standard NCP design has Thread features on the SoC and runs the application layer on a host processor, which is typically more capable (but has greater power demands) than the OpenThread device. Communication between the NCP and the host processor is managed by wpantund through a serial interface, typically using SPI or UART, over the Spinel protocol. The benefit of this design is that the higher-power host can sleep while the lower-power OpenThread device remains active to maintain its place in the Thread network. And since the SoC is not tied to the application layer, development and testing of applications is independent of the OpenThread build.

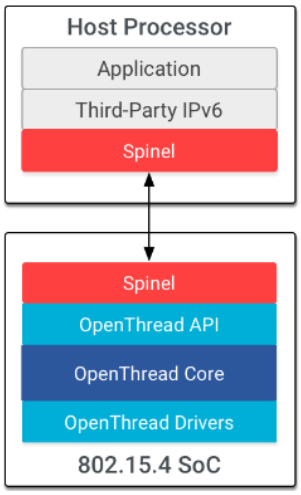


Figure 9 NCP

3.4. OpenThread Border Router

A OpenThread Border Router (OTBR) connects a Thread network to other IP-based networks such as Wi-Fi or Ethernet. To communicate outside the mesh network a Border Router is needed. The OTBR routes packets between your local network and the Thread mesh network. It does not look at the content of these packets, it just forwards them.

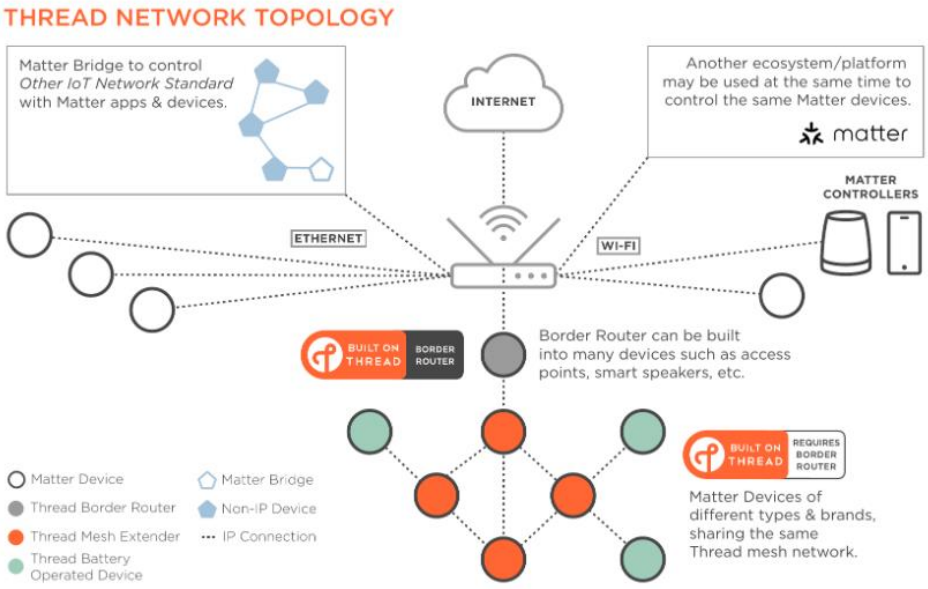


Figure 10 Thread network topology

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

3.5. Rights – OpenThread – Thread

OpenThread released by Google is an open-source implementation of Thread technology. If a company uses OpenThread to build a product, they need to be a member of the Thread Group in order to gain the Intellectual Property (IP) rights to ship Thread products and to complete product certification, which ensures that products using Thread work together effortlessly and securely right out of the box. If developers choose not to join Thread Group and ship products using Thread technology, they are not conferred the IP rights required to practice and ship Thread technology, and may subject themselves to legal action, including but not limited to licensing fees.

4. Raspberry Pi 4

In this project, a Raspberry Pi 4 is used as a Single Board Computer. It not only runs the Breakout game but also functions as a Matter hub using the CHIP tool and as a OpenThread Border Router.

Figure 11 Raspberry Pi 4

The Raspberry Pi 4 is a single-board computer developed by the Raspberry Pi Foundation. It is a popular choice for hobbyists, educators, and professionals due to its affordability, versatility, and ease of use. It is powerful enough for this use case, the most important specs for us is:

- **CPU:** Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
- **RAM:** 4GB LPDDR4-3200 SDRAM
- **Storage:** Micro-SD card slot for loading operating system and data storage
- **Connectivity:**
    - **USB Ports:** It has two USB 3.0 ports and two USB 2.0 ports
    - **Ethernet:** Gigabit Ethernet (RJ45) port for wired networking
    - **Wireless:** Dual-band 802.11ac wireless networking and Bluetooth 5.0
- **Display output:** It has two micro-HDMI ports

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

4.1. Radio Co-Processor

The OpenThread Border Router (OTBR) must have physical access to the IEEE 802.15.4 network that is used by the Thread protocol. Raspberry Pi does not have such radio capability, we need to add a Radio Co-Processor to achieve this.

As Radio Co-Processor I used a wireless starter kit mainboard (BRD4001A) developed by Silicon Labs with a EFR32xG21 SoC plug-in board (BRD4181B). This starter kit provides all the necessary features to set up a OTBR.

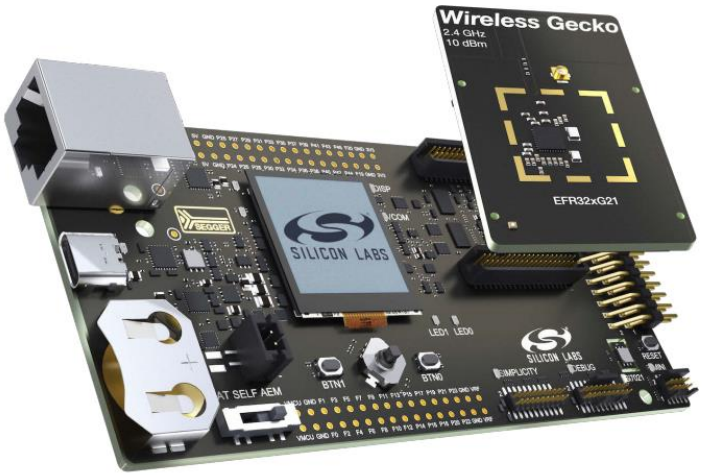


Figure 12 EFR32xG21 RCP

BRD4181B RADIO BOARD FEATURES

- EFR32xG21 Wireless Gecko Wireless SoC with 1024 kB Flash and 96 kB RAM (EFR32MG21A010F1024IM32).
- Inverted-F PCB antenna (2.4 GHz band)
- 2x user color LEDs (red and green)

MAINBOARD FEATURES

- Advanced Energy Monitor
- Packet Trace Interface
- Logic analyzer (BRD4002A only)
- Virtual COM port
- SEGGER J-Link on-board debugger
- External device debugging
- Ethernet and USB connectivity
- Low-power 128x128 pixel Memory LCD-TFT
- User LEDs / pushbuttons
- Joystick (BRD4002A only)
- 20-pin 2.54 mm EXP header
- Breakout pads for Wireless SoC I/O
- CR2032 coin cell battery support

SOFTWARE SUPPORT

- Simplicity Studio
- Energy Profiler
- Network Analyzer

ORDERING INFORMATION

- SLWSTK6006A
- SLWSTK6023A
- SLWRB4181B

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5. Matter protocol

5.1. Terms

In order to understand this section we need to know the definitions of several terms.

Term	Definition
Administrator	A Node having Administer privilege over at least the Access Control Cluster of another Node.
Attribute	A data entity which represents a physical quantity or state. This data is communicated to other Nodes using commands.
Border Router	A router, also known as Edge Router, that provides routing services between two IP subnets (typically, between a hub network and a peripheral network).
Bridge	A Node that represents one or more non-Matter devices on the Fabric.
Bridged Device	A non-Matter device that is represented on the Fabric by a Bridge so it can be used by Nodes on the Fabric
Certificate Authority (CA)	An entity that issues digital certificates such as a DAC or NOC.
Client	A Cluster interface that typically sends commands that manipulate the attributes on the corresponding server cluster. A client cluster communicates with a corresponding remote server cluster with the same cluster identifier.
Cluster	A specification defining one or more attributes, commands, behaviors and dependencies, that supports an independent utility or application function. The term may also be used for an implementation or instance of such a specification on an endpoint.
Command	Requests for action on a value with an expected response which may have parameters and a response with a status and parameters.
Commission	To bring a Node into a Fabric.
Commissioner	A Role of a Node that performs Commissioning.
Commissionee	An entity that is being Commissioned to become a Node.
Commissioning	Sequence of operations to bring a Node into a Fabric by assigning an Operational Node ID and Node Operational credentials.
Controller	A Role of a Node that has permissions to enable it to control one or more Nodes.
Controlee	A Role of a Node that has permissions defined to enable it to be controlled by one or more Nodes.
Device	A piece of equipment containing one or more Nodes.
Device Attestation Certificate (DAC)	An RFC 5280 compliant X.509 v3 document with attestable attributes.
Discriminator	A 12-bit value used to discern between multiple commissionable Matter device advertisements.
Endpoint	A particular component within a Node that is individually addressable.
Fabric	A logical collection of communicating Nodes, sharing a common root of trust, and a common distributed configuration state.
Node	An addressable entity which supports the Matter protocol stack and (once Commissioned) has its own Operational Node ID and Node Operational credentials. A Device MAY host multiple Nodes.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Node Operational Certificates (NOCs)	are installed during the Matter network commissioning by the commissioner together with Trusted Root CA Certificates.
Onboarding Payload	The information needed to start the process of commissioning a Device.
OTA Provider	A Node implementing the OTA Software Update Provider role.
OTA Requestor	A Node implementing the OTA Software Update Requestor role.
Product Attestation Authority	An entity which operates a root level Certificate Authority for the purpose of Device Attestation.
Product ID (PID)	A 16-bit number that identifies the type of a Device, uniquely among the product types made by a given vendor.
Root of trust	a concept within Matter that is centered around a certification authority (CA), identified by Root Public Key (Root PK).
Server	A Cluster interface that typically supports all or most of the attributes of the Cluster. A Server Cluster communicates with a corresponding remote Client Cluster with the same Cluster identifier.
Vendor ID (VID)	A 16-bit number that uniquely identifies the Vendor of the Device.

Figure 13 Tabel Matter terms

5.2. About

Matter is a royalty-free connectivity standard for smart home devices that was announced in December 2019 and officially released in September 2022. It is designed to enable seamless communication and interoperability between smart home devices from different manufacturers, regardless of their underlying technology or platform. Matter was developed by the Connectivity Standards Alliance (CSA), a non-profit organization that promotes interoperability among diverse technology ecosystems. The CSA was formed in 2021 by merging the Zigbee Alliance and the Project Connected Home over IP (CHIP) working group.

5.3. Documents

The Matter protocol is specified in Three main documents and can be downloaded from the CSA website. These documents are necessary when developing a Matter product. With every update these documents changes so make sure when developing a product you get the latest documents.

- **Matter-(version)-Core-Specification**
  In this document you can find all about the features of the protocol, the requirements, technical specifications etc.

- **Matter-(version)-Device-Library-Specification**
  This document tells which device-types are included in this specific version, it also tells what clusters can and must be used with a specific device-type.

- **Matter-(version)-Application-Cluster-Specification**
  This document tells all the available clusters in this specific version. It tells what attributes and commands are part of a clusters and which attributes and commands are mandatory or optional for a specific cluster.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.4. Architecture

5.4.1. Network

Matter is a universal IPv6-based communication protocol designed for smart home and the Internet of Things devices. Matter works on the Application Layer of the OSI model.

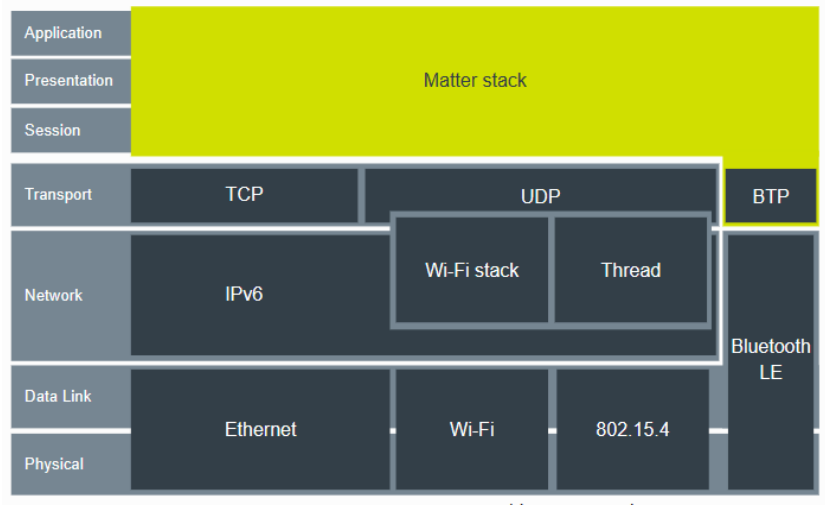


Figure 14 Matter stack

The Matter protocol stack is divided into layers to separate the different responsibilities and introduce a good level of encapsulation amongst the various parts of the protocol stack.

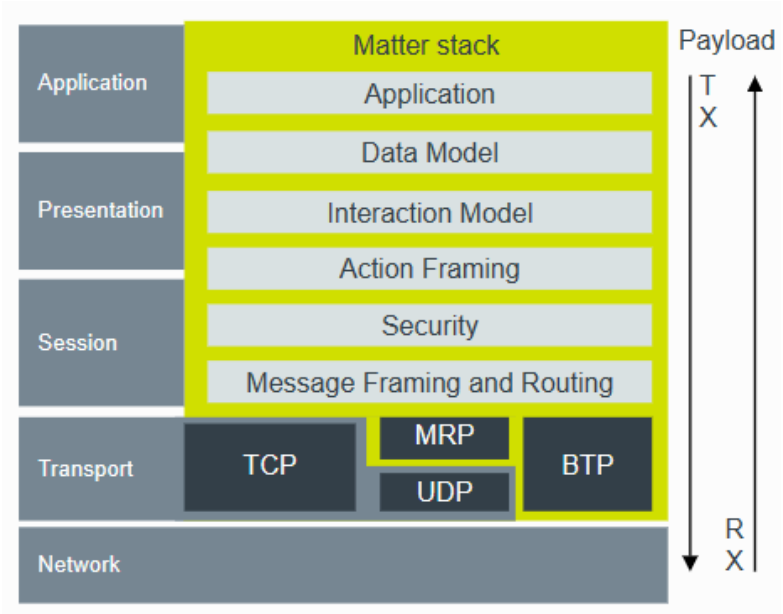


Figure 15 Matter stack layers

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

- **Application Layer:** High-order business logic of a device. For example, an application that is focused on lighting might contain logic to handle turning on/off the light as well as its color characteristics.
- **Data Model:** The data layer corresponds to the data and verb elements that help support the functionality of the application. The Application operates on these data structures when there is an intent to interact with the device.
- **Interaction Model:** The Interaction Model layer defines a set of interactions that can be performed between a client and server device. For example, reading or writing attributes on a server device would correspond to application behavior on the device. These interactions operate on the elements defined at the data model layer.
- **Action Framing:** Once an action is constructed using the Interaction Model, it is serialized into a prescribed packed binary format to encode for network transmission.
- **Security:** An encoded action frame is then sent down to the Security Layer to encrypt and sign the payload to ensure that data is secured and authenticated by both sender and receiver of a packet.
- **Message Framing & Routing:** With an interaction encrypted and signed, the Message Layer constructs the payload format with required and optional header fields; which specify the message's properties and some routing information.
- **IP Framing & Transport Management:** After the final payload has been constructed, it is sent to the underlying transport protocol for IP management of the data.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.4.2. Network Topology

In principle, any IPv6 network is suitable for Matter deployment butt the focus is on three link layer technologies: Ethernet, Wi-Fi and Thread. Matter treats networks as shared resources: it makes no stipulation of exclusive network ownership or access. As a result, it is possible to overlay multiple Matter networks over the same set of constituent IP networks.

This protocol may operate in the absence of globally routable IPv6 infrastructure. This requirement enables operation in a network disconnected or firewalled from the global Internet. It also enables deployment in situations where the Internet Service Provider either does not support IPv6 on consumer premises or where the support proves otherwise limiting.

This protocol supports local communications spanning one or more IPv6 subnets. Canonical networks supporting a fabric may include a Wi-Fi/Ethernet subnet, or one or more low power and lossy network (LLN) subnets. In this version of the specification, Thread is the supported LLN standard.

Single network

In the single network topology, all Matter devices are connected to a single network. This could by a Thread network, Wi-Fi or Ethernet network. In the case of Wi-Fi/Ethernet, the network could in fact span multiple Wi-Fi and/or Ethernet segments provided that all the segments are bridged at the link layer.

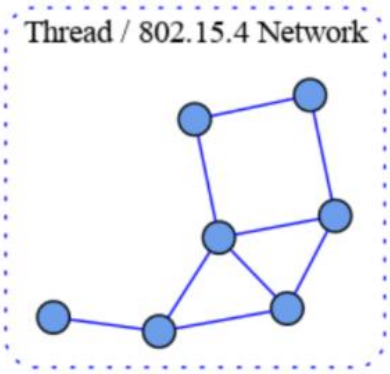


Figure 17 Single network topology Thread

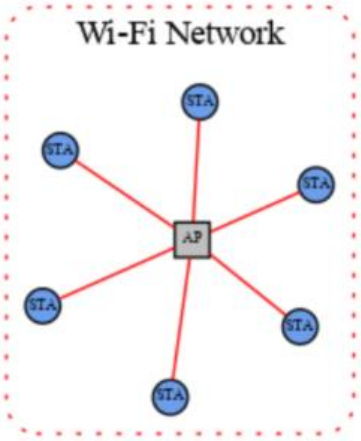


Figure 16 Single network topology Wi-Fi

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Star network topology

The star network topology consists of multiple peripheral networks joined together by a single hub network. The hub network will be the customer's home network. while the peripheral networks can be of any supported network type. A peripheral network must always be joined directly to the hub network via one or more Border Routers.

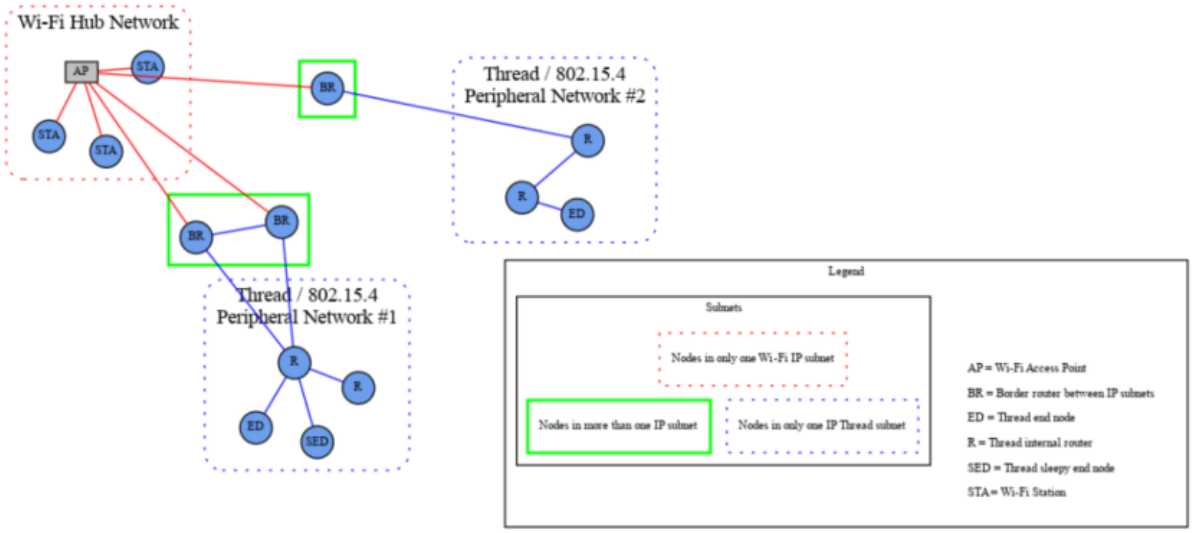


Figure 18 Star network topology Thread

In the star network topology any number of peripheral network may be present in a single fabric, including networks of the same type. Nodes may have interfaces onto any network and can communicate directly to other nodes on the same network. If a node needs to cross a network to communicate it must flow through a Border Router.

Regardless of the Network topology being used, Matter has a concept of Fabrics. A Matter Fabric is a security domain that contains a collection of nodes. These nodes can be identified and can communicate with each other within the context of that security domain. Each Matter Fabric has a unique Node ID for each node within the fabric and has a unique Fabric ID. Any Matter device can be a part of multiple Matter fabrics, and in turn will have multiple associated Node IDs / Fabric IDs depending on the fabric it is communicating with.

This protocol places a set of requirements on the Border Router. These requirements pertain to address assignment, route assignment and advertisement, multicast support, and discovery proxying.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.5. Data Model

The data model in Matter is a hierarchical modeling of a devices features, including nodes, endpoints, clusters and device types where the node is the highest level data element. A single device can be represented by one or more nodes. An environment where multiple Matter nodes interoperate is referred as a Matter fabric.

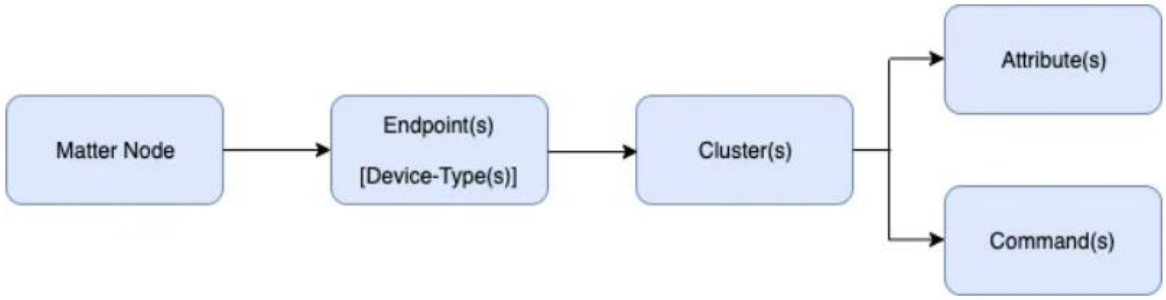


Figure 19 Data Model Hierarchical

5.5.1. Node

This is a uniquely network addressable entity that exposes some functionality. This is typically a physical device that a user can recognize as a whole device. The role of a node is a set of related behaviors. A node can contain one or more roles including:

- Commissioner: Refers to the process of assigning Fabric credentials to a new device.
- Controller: A node that can control one or more nodes such as a On/Off switch
- Controlee: A node that can be controlled by one or more nodes. Such as an actor. Devices that have the controller role cannot be a controlee.
- OTA Provider: Provides OTA software updates.
- OTA Requestor: Requests OTA software updates.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

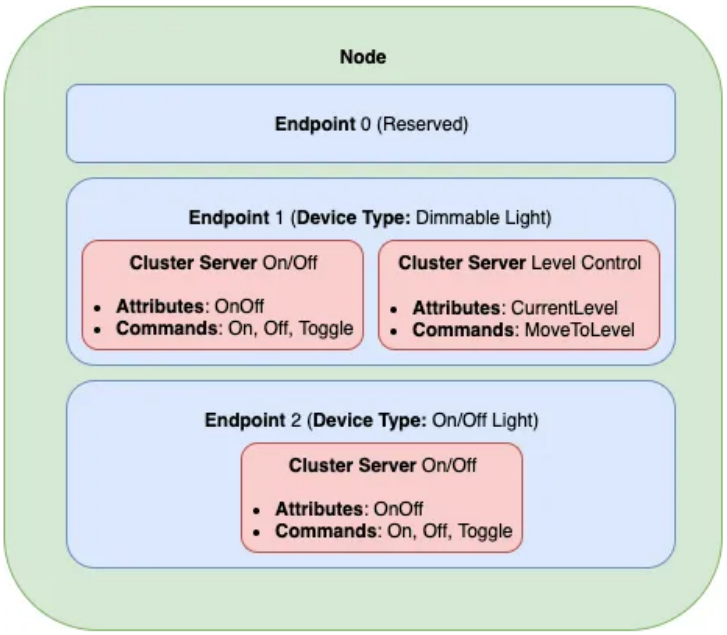


Figure 20 Node explanation

5.5.2. Endpoint

Each node has one or more endpoints. A endpoint contain a set functionality's of a single device. In the example above endpoint 1 is a dimmable light that have the functionality turning on or off and have a functionality level control, that controls the brightness of the light. Endpoint 2 have only the functionality turning on or off. Note that endpoint 0 is reserved for utility functions like diagnostics, etc.

5.5.3. Clusters

A cluster groups together commonly used functionality in a reusable building block. Endpoint 1 has two clusters which describes their own functionality. Within the clusters they contain:

- **Attributes**: Attributes are data entities that represent a physical quantity or state. Each attribute is listed in a table with data quality columns: ID, Name, (Data) Type, Constraint, other Quality, Access, Default (value), and Conformance. An attribute also defines its associated semantics and behavior.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Table 12. Attributes of the On/Off Server Cluster

ID	Name	Type	Constraint	Quality	Default	Access	Conformance
0x0000	OnOff	bool	all	SN	FALSE	R V	M
0x4000	GlobalSceneControl	bool	all		TRUE	R V	LT
0x4001	OnTime	uint16	all	X	0	RW VO	LT
0x4002	OffWaitTime	uint16	all	X	0	RW VO	LT
0x4003	StartUpOnOff	StartUpOnOffEnum	desc	XN	MS	RW VM	LT

Figure 21 Attributes On/Off Server Cluster

- **Commands**: A cluster command provides an ability to invoke a specific behavior on the cluster. A command may have parameters that are associated with it. Each command SHALL be listed in a table with data quality columns: ID, Name, Direction, Response, Access, Conformance. The command table SHALL define the direction of the command as either client to server or server to client. The command table SHALL define the access privileges for each request command or omit the privileges for the default.

Table 13. Command IDs for the On/Off Cluster

ID	Name	Direction	Response	Access	Conformance
0x00	Off	client ⇒ server	Y	O	M
0x01	On	client ⇒ server	Y	O	M
0x02	Toggle	client ⇒ server	Y	O	M
0x40	OffWithEffect	client ⇒ server	Y	O	LT
0x41	OnWithRecallGlobalScene	client ⇒ server	Y	O	LT
0x42	OnWithTimedOff	client ⇒ server	Y	O	LT

Figure 22 Commands On/Off Cluster

- **Events:** Events are a type of attributes that communicate device state changes. They can also be treated as historical data records of something that happened on the device in the past.

The clusters that are supported can be found at the **Matter-(version)-Application-Cluster-Specification** document.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.5.4. Custom Cluster

It is possible to create a custom cluster that is suitable for a specific use case. For example, for a project you need a specific attribute that is not available in the cluster that you going to use. You can extend that cluster so the attribute is available. While creating a custom cluster is possible, the cluster will only work between your device and the custom Matter controller expecting the cluster. The custom cluster will not work with a non-custom Matter controller. For interoperability, the cluster needs to be published in the cluster specifications, which only occurs if your organization is at least an adoption-level member of the CSA.

If considering working with a custom cluster, I recommend reading **section 7.10.4** in the core specification.

5.5.5. Cluster Client / Server

A cluster server is stateful and holds attributes, events and commands while a client is stateless and is responsible to initiate interactions with a cluster server.

5.5.6. Device Type

A Matter device type is an officially defined collection of requirements for one or more endpoints. Device types are intended to ensure interoperability of different device brands on the market. Each device type definition is composed of the following elements:

- Device type ID
- Device type revision
- One or more mandatory clusters, including each cluster's minimum revision

All the device types are defined in the **Matter-(version)-Application-Cluster-Specification.pdf** document.

Relating Matter to Zigbee

Ultimately, Matter serves to extend existing protocol stacks to maintain and bolster their architecture for future use. Thus, the Data Model originates from and resembles the Dotdot Architecture Model and Chapter 2 of the Zigbee Cluster Library Specification found here: https://csa-iot.org/developer-resource/specifications-download-request/. The Matter Data Model better defines the architecture in the Zigbee Cluster Library while keeping the certifiable cluster specifications.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.6. Interaction Model

The Matter Interaction Model defines the methods of communication between nodes and serves as the common language for node to node transmission. Node communicate with each other through interactions. Interactions are a sequence of one or more transactions, which in turn are a sequence of actions.

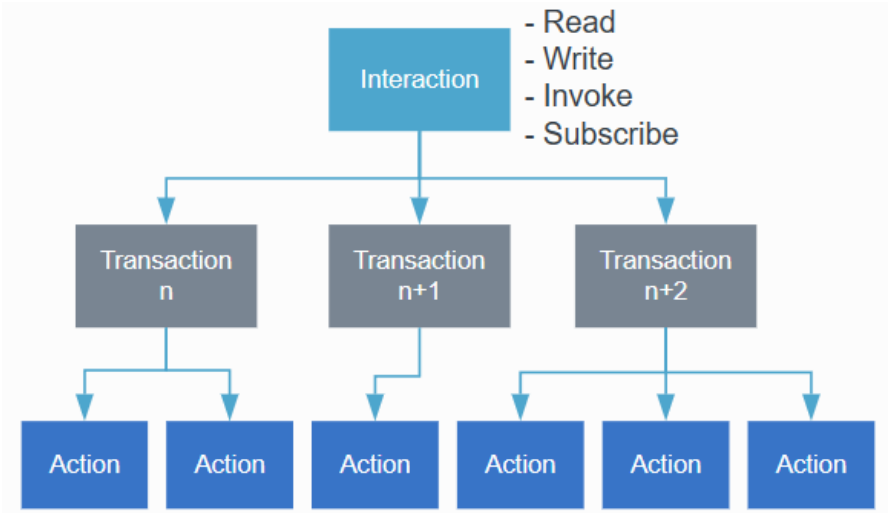


Figure 23 Interaction Model

For example, when a client cluster initiate a Read Transaction and requests to read an attribute, the server cluster can respond by giving the attribute. The client request and the server response are separated actions but they are part of the same Read Transaction, these actions and transactions belong to the Read Interaction.

There are four types of interactions:

- **Read Interaction**
- **Write Interaction**
- **Subscribe Interaction**
- **Invoke Interaction**

These types except for Subscribe Interaction consist of one transaction. The Interaction Model supports five types of transactions:

- **Read**: Get attributes and/or events from a server.
- **Write**: Modify attribute values.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

- **Invoke**: Invoke cluster commands.
- **Subscribe**: Create subscription for clients to receive periodic updates from servers automatically.
- **Report**: Maintain the subscription for the Subscribe Interaction.

There are several concepts that is important for understanding transactions before explaining the different Interactions.

- **Initiators and Targets**: Interactions happen between initiator and target nodes. The initiator starts the transaction and the target responds. Usually the client cluster is the initiator node and de server the target node.
- **Transaction ID**: The transaction ID field must be present in all actions that are part of a transaction to indicate the logical grouping of the actions as part of one transaction. All actions that are part of the same transaction must have the same transaction ID.
- **Groups**: Groups allow an initiator to send an action to multiple targets. This type of communication is known as a groupcast, which leverages IPv6 multicast messages.
- **Paths**: Paths are the location of the attribute, event or command an interaction seeks to access. For example:
  - `<path> = <node> <endpoint> <cluster> <attribute / event / command>`
  - `<path> = <group ID> <attribute / event / command>`

  When groupcasting, a path may include the group or a wildcard operator to address several nodes simultaneously, decreasing the number of actions and thus decreasing the response time of an interaction. Without groupcasting there is a chance of latency between multiple devices reacting to an interaction.

5.6.1. Read Interaction

When an initiator wishes to determine the value of one or more attributes or events on a node, an Read Interaction is generated. In this interaction the following actions occur:

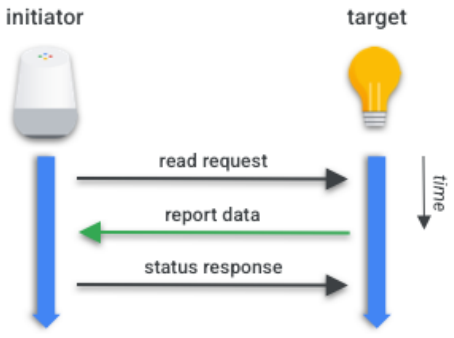


Figure 24 Read Transaction

- **Read Request:** First action of a Read transaction. The initiator requests a list of

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

the target's attributes and/or events.

- **Report Data:** Generated in response to the Read Request action. The target sends the requested list of attributes and/or events back along with a suppress response and a subscription ID.
    - Suppress response: a flag that determines whether the status response to this action should be suppressed.
    - Subscription ID: Integer that identifies the subscription transaction, only included if the report is part of a Subscription Transaction.
- **Status Response Action (optional):** Only generated if the suppress response flag is not set. A status Response Action will be generated with a status code:
    - SUCCESS: to continue the interaction
    - INVALID SUBSCRIBTION: if the action is part of a Subscribe interaction and the Subscription ID is invalid
    - FAILURE: to terminate the interaction

Read Transactions are restricted to unicast, the Read Request and Report Data actions cannot target groups of nodes because the Status Response Action cannot be generated as a response to a groupcast.

5.6.2. Subscription Interaction

Subscription is used by a initiator to automatically receive updates of an attribute or event. This creates a relationship between two nodes where the target/publisher periodically generates Report Data Actions to the initiator/subscriber.

A subscribe Request Action contains the following elements:

- **Min Interval Floor:** The minimum interval between reports
- **Max Interval Ceiling:** The maximum interval between reports
- **Attribute Reports:** A list of zero or more of the reported Attributes requested in the Read Action

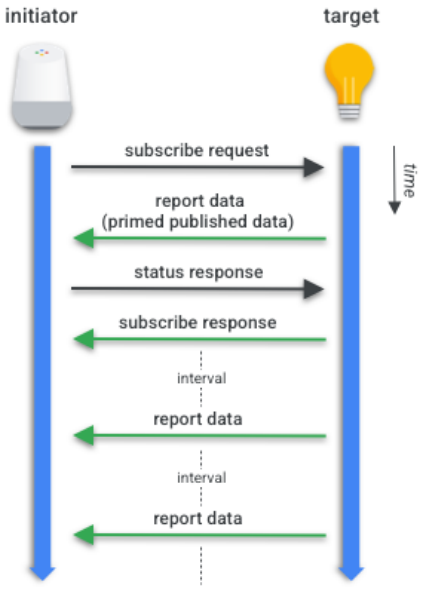


Figure 25 Subscription Transaction

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Request.

- **Event Reports:** A list of zero or more reported Events.

After the Subscribe request by the initiator, the target responds with a Report Data Action containing the first batch of reported data, the primed published data. Then the initiator acknowledges with a Status Response Action send to the target. Once the Target receives a Status Response Action reporting no errors, it sends a Subscribe Response Action. Then the target will send Report Data Action periodically at the negotiated interval until the subscription is lost or cancelled.

Subscription interactions have some restrictions:

- The subscribe Request and Response Action are Unicast-only
- Report Data Actions in the same Subscription Interaction must have the same Subscription ID.
- If the subscriber does not receiver a Report Data Action within the maximum interval between Actions, the subscription will be terminated.

5.6.3. Write Interaction

The Write Interaction is generated when a initiator wants to modify values of one or more attributes located on one or more nodes. The initiator has the option to use a Timed or a Untimed Write Transaction.

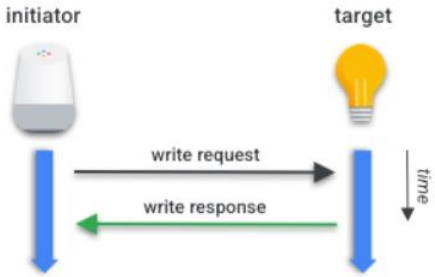


Figure 26 Write Transaction

Untimed Write Transaction

In the Untimed Write Transaction there are two actions:

- **Write Request Action:**
  A Write Transaction works similar to the Read Request Action. The initiator provides the target with:
  - **Write Request:** A list of one or more tuples containing Path and data.
  - **Timed Request:** A flag that indicates whether this action is part of a Timed Write Transaction.
  - **Suppress Response:** A flag that indicates whether the Response Status Action should be suppressed
- **Write Response Action:**
  When the target receives the Write Request Action it will respond with the Write Response Action containing:

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

- **Write Response:** A list of paths and error codes every Write Request send on the Write Request Action.

The Write Request Action may be a groupcast, but in this case the Suppress Response flag must be set. The rationale is that otherwise the network might be flooded by simultaneous responses from every member of a group. To enable this behavior, the Path used in the Write Requests list may contain Groups and alternatively they may contain wildcards, but only on the Endpoint field.

Timed Write Transaction

In this transaction there where added a few steps to the untimed write transaction.

- **Timed Requests Action:**
  When the initiator starting a transaction, the action contains:
  - **Timeout:** how many milliseconds this transaction may remain open. During this period the next action sent by the Initiator will be considered valid.

When the target receives the Timed Request Action, he must acknowledge with a Status Response Action. When the initiator receives a response with no errors, it will send a Write Request Action.

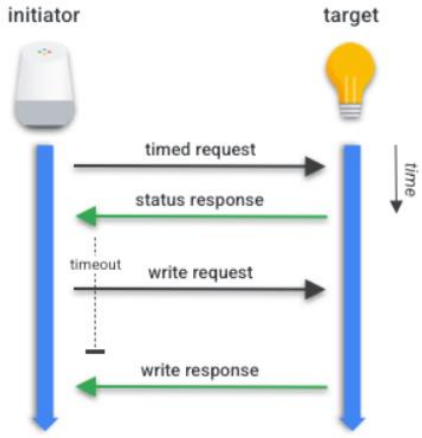


Figure 27 Timed Write Transaction

The rest of the actions are the same as the Untimed Write Transaction.

5.6.4. Invoke Interaction

This interaction is used for invoking one or more cluster commands on a target. It is a similar way of the write transactions, Invoke transactions support also timed and untimed transactions.

Untimed Invoke Transaction

Like in the untimed write transactions, the untimed invoke transactions contains two actions:

- **Invoke Request Action:**
  Similar to the Read Request Action and Write Request Action, in this Action the Initiator provides the Target with:

Figure 28 Untimed Invoke Transaction

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

- **Invoke Requests:** a list of paths to Cluster Commands, as well as optional arguments to the commands, named Command Fields.
- **Timed Request:** a flag that indicates whether this action is part of a Timed Invoke Transaction.
- **Suppress Response:** a flag that indicates whether the Invoke Response Action should be suppressed.
- **Interaction ID**: an integer used for matching the Invoke Request Action to the Invoke Response Action.

- **Invoke Response Action:**
  After the target receives the invoke request action it will respond with an invoke response action that carries:
  - **Invoke Responses:** a list of command responses or status for every invoke request sent.
  - **Interaction ID:** a integer used for matching the Invoke Response Action to the Invoke Request Action.

Timed Invoke Transaction

Similar to the timed write transaction, the timed invoke transaction added the following step.

- **Timed Request Action:**
  A Initiator starts the Transaction sending this Action that contains:
  - **Timeout:** how many milliseconds this transaction may remain open. During this period the next action sent by the Initiator will be considered valid.

Once the Timed Request Action is received, the Target must acknowledge the Timed Request Action with a Status Response Action. Once the Initiator receives a Status Response Action reporting no errors, it will send a Invoke Request Action.

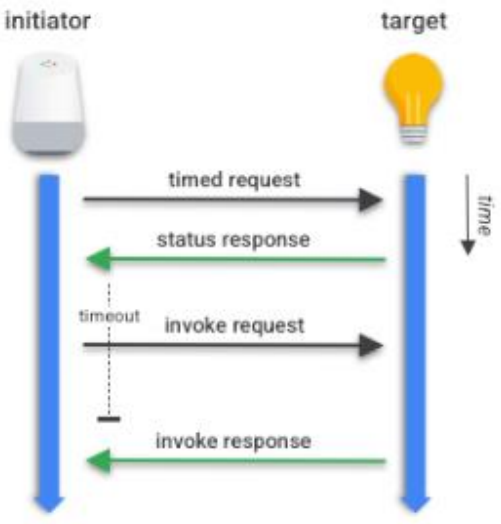


Figure 29 Timed Invoke Transaction

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.7. Security

To protect the Matter fabric, there are several security features implemented to make sure that only trustworthy device can join the network and protecting the messages that are exchanged between the fabric nodes.

5.7.1. Session establishment

This is used the exchange encryption keys that are required for a secure communication between nodes. It also involves mutual node authentication, which assures both nodes that they initiate communication with a trusted peer.

There are two session establishment methods available:

- **PASE**: Passcode-Authenticated Session Establishment
  When using PASE, both nodes share the same secret in the form of 8-digit passcode. The shared secret is used by the SPAKE2+ algorithm to ensure a safe exchange of keys over non-secure channel. This process takes place when commissioning the device.

- **CASE**: Certificate-Authenticated Session Establishment
  When using CASE, both nodes own Node Operational Certificates that chain back to the same root of trust. The NOCs are used by the SIGMA algorithm to ensure a mutual node authentication and a safe exchange of keys over non-secure channel. This process takes place while establishing the secured communication between nodes that are already commissioned.

Root of trust is a concept within Matter that is centered around a certification authority (CA), identified by Root Public Key (Root PK). The CA is a device tasked with issuing and assigning Node Operational Certificates (NOCs) or Intermediate Certificate Authority Certificates (ICACs). NOCs are installed during the Matter network commissioning by the commissioner together with Trusted Root CA Certificates.

5.7.2. Message confidentiality and integrity

After exchanging the keys and establishing secure channel, the 128-bit AES-CCM algorithm is used to provide both confidentiality and integrity of exchanged messages.

A Matter message consists of the following elements:

- **Message Header:** Carries session and transport-related information.

- **Protocol Header:** Describes semantics of the message.

- **Payload:** Actual protocol-specific content of the message.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

While the AES-CCM algorithm ensures the integrity of all three elements, only Protocol Header and Payload get encrypted. This is because Message Header contains fields, such as Security Flags and Message Counter, which are used to calculate the AES-CCM Nonce that is necessary to decrypt the remaining part of a message.

5.8. Commissioning

When we want to add a device to a Matter network, we first need to commission it. You can think of it as the initial pairing. Although Matter communication in general uses Wi-Fi, Ethernet and Thread, commissioning can be performed over Bluetooth Low Energy (BLE).

The commissioning process takes place between a commissioner and a commissionee. Where the commissioner is the controller that is carrying out of the commissioning and the commissionee is the device that wants to join the fabric.

5.8.1. Prerequisites

The controller must get onboarding payload from the commissionee to start the procedure. The data payload includes the following information, among other things:

- 16-bit Vendor ID and 16-bit Product ID
- 12-bit device discriminator
- 27-bit setup passcode
- 8-bit Discovery Capabilities Bitmask

5.8.2. Onboarding payload

The Onboarding Payload is the information used by the Commissioner to ensure interoperability between commissioners and devices. The commissioner can collect this information in different ways:

- **QR Code:** which you can scan using a mobile device with the ecosystem of your choice.
- **QR Code Payload:** which is an alphanumeric code that you can use in command-line tools. For testing purposes, it can be printed to the UART console or be shared using an NFC tag. This code is represented visually by the QR Code.
- **Manual Pairing Code:** provides the onboarding information as a sequence of digits that can be used with most Matter commissioners

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.8.3. Commissioning steps

The commissioning process consists of steps shown in the figure below.

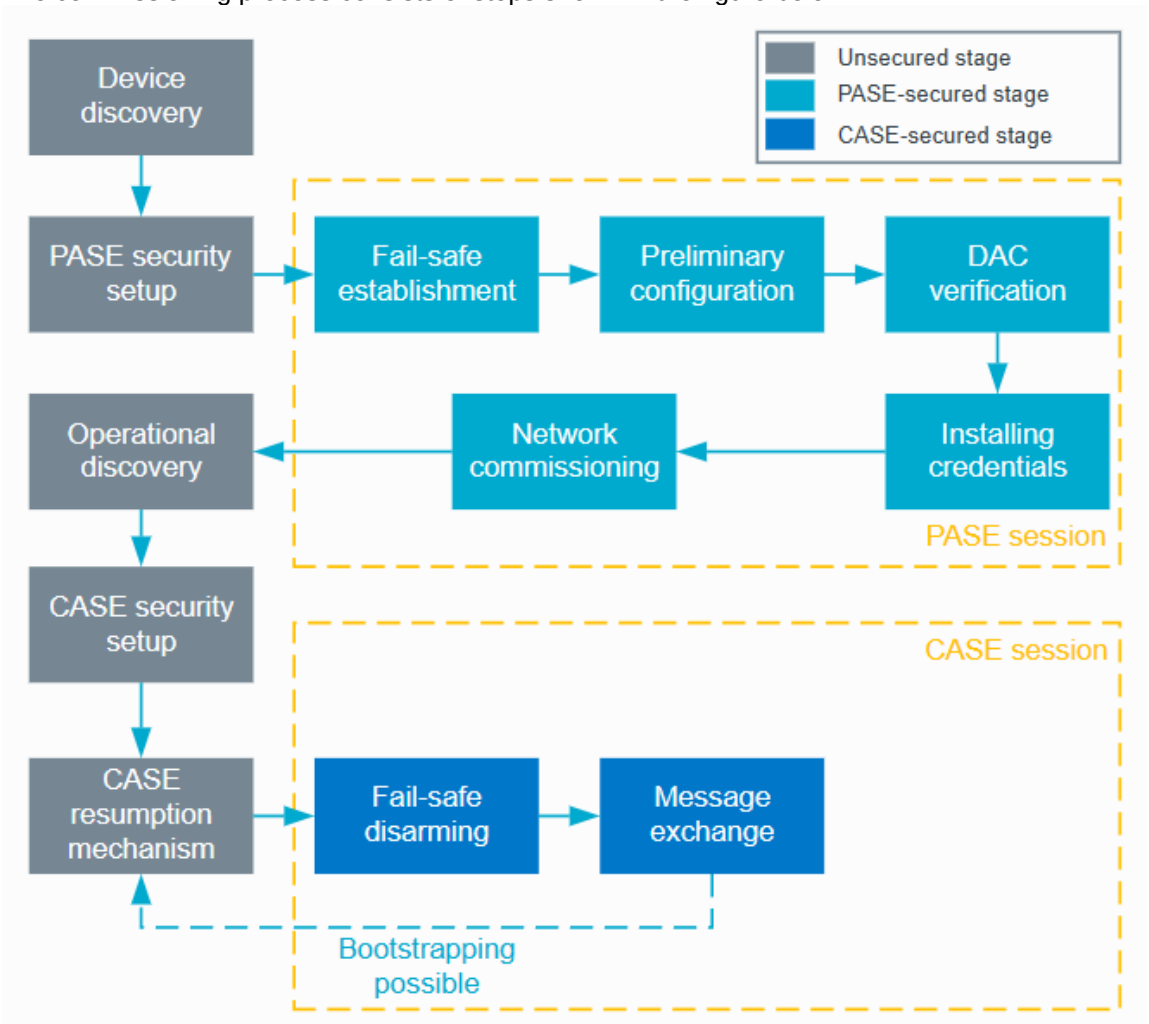


Figure 30 Commissioning steps

Device discovery

The commissioner discovers devices that can be commissioned onto the network. The commissionee needs to advertise their presence so the commissioner knows about their existence. This can happen using the following methods:

- **Bluetooth Low Energy:** This method is used especially if the node is being added to its first Matter fabric.
- **DNS-SD:** This method is commonly used if the node is connected to Ethernet or is already a member of a Wi-Fi or Thread network.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

PASE security setup

The commissioner runs the Passcode-Authenticated Session Establishment (PASE) protocol, which is exclusive to the commissioning process. This protocol is used to establish the first session between devices that take part in commissioning. The session is established with a passcode provided out-of-band and that is used to derive encryption keys. This passcode is known only to the commissioner and the commissionee. There can be only one ongoing PASE sessions at a time.

Fail-safe establishment

The commissioner requests the commissionee to back up its original configuration. The fail-safe acts as a back-up, but it also starts a timer that sets a limit for the whole commissioning process. The timer is disabled with the disarming of the fail-safe at the end of commissioning.

Preliminary configuration

The commissioner reads the Basic Information Cluster attributes of the commissionee and its device type. It then configures the commissionee with regulatory information, such as location and country, and the current UTC time.

Device Attestation Certificate verification

The commissioner checks whether the commissionee is a certified Matter device. As part of this verification, the commissioner generates a random 32-bit attestation nonce and sends it to the commissionee, who should return the signed attestation information that includes the nonce. Usage of a nonce prevents replay attacks against commissioners. The commissioner then validates the attestation information.

The verification succeeds if the device is able to prove the validity and ownership of the mandatory Matter Device Attestation elements. If the validity and ownership cannot be proven, the verification fails. The commissioner can then either terminate or continue the commissioning procedure.

Installing credentials

The commissioner installs Node Operational Certificate (NOC) and Operational ID on the commissionee. The commissionee becomes the new node of the Matter fabric. The node is identified by a tuple consisting of the Root PK, Fabric ID, and Node ID. (While the fabric is identified by a tuple consisting of the Root PK and the Fabric ID.)

Network commissioning

The commissioner provisions the commissionee node with the operational network credentials, either Wi-Fi or Thread, and requests the commissionee to connect to the network.

Operational discovery

The commissioner discovers the commissionee node on the operational network using DNS-SD. This way, the commissioner learns the IP address of the node.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

CASE security setup

The commissioner and the node use the Certificate-Authenticated Session Establishment (CASE) protocol to establish secure communication. The CASE protocol is in charge of exchanging NOCs to set up a session secured with a new pair of keys. The CASE connection is reset each time a device breaks the connection.

Fail-safe disarming

The commissioner requests the commissionee node to remove the stored configuration backup. This also stops the fail-safe timer.

Message exchange

The commissioner and the commissionee start exchanging AES-encrypted messages on the operational network.

5.9. Fabric

Whenever a set of Devices in a network shares the same security domain, and thus allows secure communication between Nodes, this set is called a Fabric. Fabrics share the same Certificate Authority (CA) top-level certificate (Root of Trust) and within the context of the CA, a unique 64-bit identifier named Fabric ID. Thus the commissioning process is the assignment of the Fabric credentials to a new Node so it may communicate with other Nodes in the same Fabric.

5.9.1. Multi-Admin

Nodes may also be commissioned on more than one Fabric. This property is often referred to as multi-admin. For instance, we may have a Device commissioned to both the manufacturer's Fabric and a cloud ecosystem's Fabric, with each Fabric handling a different set of encrypted communications and operating independently.

As several Fabrics may coexist, a Device might have several sets of Node operational credentials. However, the Node's Data Model is shared: the Cluster Attributes, Events, and Actions are common between Fabrics. Thus, although Thread and/or Wi-Fi credentials are set during the commissioning process, they are part of the Networking Operational Cluster, being shared between all the Fabrics and part of the node's DM, not the Fabric credentials.

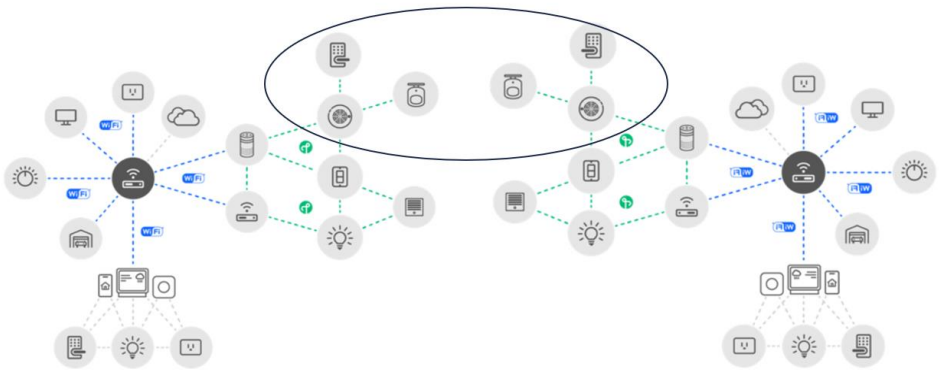


Figure 31 Multi-Admin / Multiple Fabrics

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.10. Matter Hub & Bridge

5.10.1. Hub

A Matter hub essentially serves as the controller of the network, necessary for managing your Matter devices. It typically isn't tied to a specific ecosystem, allowing users the freedom to choose their preferred ecosystem. Additionally, the Matter Hub often functions as a Thread Border Router but not always. Example of a Matter Hub: Apple HomePod Mini

Figure 32 Apple HomePod Mini

5.10.2. Bridge

A Matter bridge act as a translator, it allows non Matter devices to connect and work with Matter devices in a fabric. With a Matter bridge Zigbee and/or Z-Wave devices can work with Matter devices, this allows more flexibility in an existing smart-home. Example of a Matter Bridge: Philips Hue Bridge.

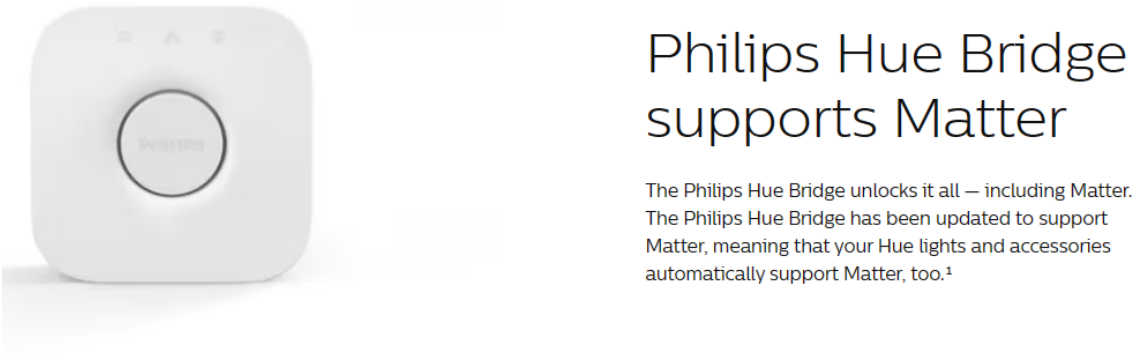


Figure 33 Philips Hue Bridge

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.11. Matter production guide

To develop a Matter end product, this topic lists the prerequisites and next steps to facilitate your production journey through Matter.

5.11.1. CSA Member

Associated membership is necessary to get a product certified by a CSA-approved testing facility. As a member you will receive membership perks like:

- Official resources to assist you in developing Matter products.
- Authorization to contribute to the Matter Github repository.
- Once approved, CSA will reserve a unique Vendor ID (VID) chosen by your organization. This VID will be needed to provision your device. Your unique VID will be added to the CSA Distributed Compliance Ledger (DCL).
- Matter Certification tool access. This allows you to evaluate your product for certification before the official certification process

Become a member at CSA Membership. You can see a list of the different memberships offered at CSA.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

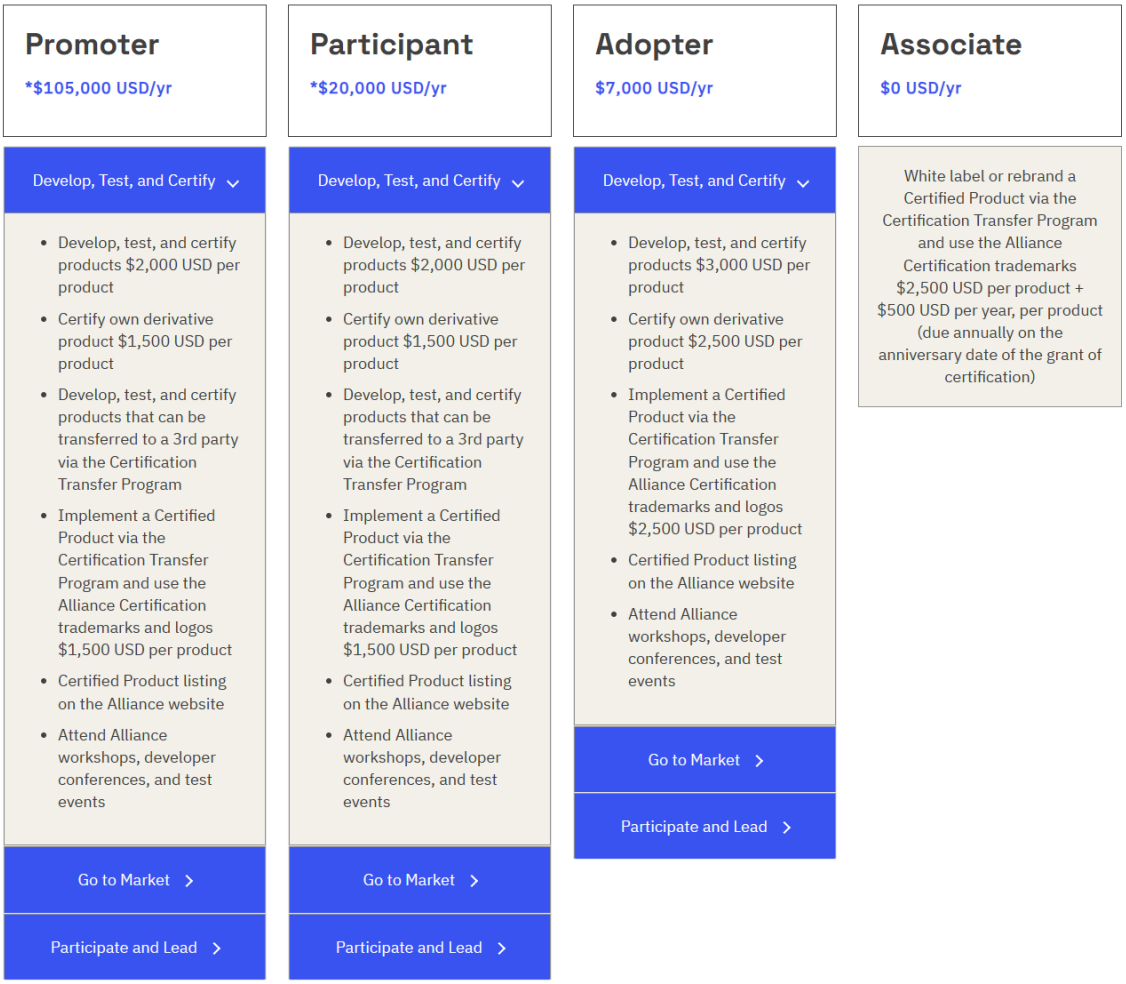


Figure 34 CSA Memberships

5.11.2. CSA Certification

Once a product is ready for production, the product must be certified by a CSA-approved testing facility to certify compliance with the Matter Standard. Check the CSA Certification process for more information. For additional resources on the certification process, refer to the Certifying your Matter Product article by Silicon Labs.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

5.12. Develop with Matter

Matter is a very complex protocol and there are different ways to develop with Matter.

5.12.1. Official Matter SDK

The CSA provides the official Matter SDK that can be find on https://github.com/project-chip/connectedhomeip. Documentation, that contains guides to develop with VS Code, to build clusters, controllers, tools, etc. can be found on https://project-chip.github.io/connectedhomeip-doc/index.html. On https://github.com/project-chip there are very useful repo's that you can use when developing.

5.12.2. Manufacturer SDK

Manufacturers like Silicon Labs, NXP, and Espressif have their own SDKs developed on top of the official SDK. Depending on which product you are going to use, you have the option to use their SDKs with VS Code or with their IDEs. Documentation is available, just search "Espressif matter SDK" for example, everything you need to know is available.

5.12.3. Certification test

Before certification you can use the certification tool available at https://github.com/project-chip/certification-tool. With this set of tools you can test your device before certification.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

6. Silicon Labs Gecko SDK V4.4.0 – SSv5

As this project uses a Silicon Labs development board (EFR32MG24), we will also be using their IDE and Software Development Kit (SDK).

The IDE used is Simplicity Studio v5. It is an Eclipse-based IDE optimized for all Silicon Labs technologies, SoCs and modules. It provides access to device specific and SDK resources, software and hardware configuration tools. It also includes a variety of advanced tools that can be used by your products.

Gecko SDK 4.4.0 is an SDK designed for developing applications for Silicon Labs' 32-bit IoT products. It integrates several wireless SDKs along with the Gecko platform into a single package. This version includes Matter version 1.2, which is the current Matter version at the time of writing.

6.1. Installation

Install the Simplicity Studio 5 IDE first. Once installed you will be asked to install the SDK's. To use Matter you need to install the Silicon Labs Matter - 2.2.0 extension (this is Matter version 1.2).

Figure 35 Gecko SDK - Silabs Matter extension

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

6.2. Matter Examples

One of the good things about the IDE and the SDK that is used is that there are several Matter examples that can be tested and modified. These can be created using the Project Wizard, selecting the target board, device and SDK will take you to the next tab, Examples. Under Wireless Technology you can select Matter and all available examples will be displayed.

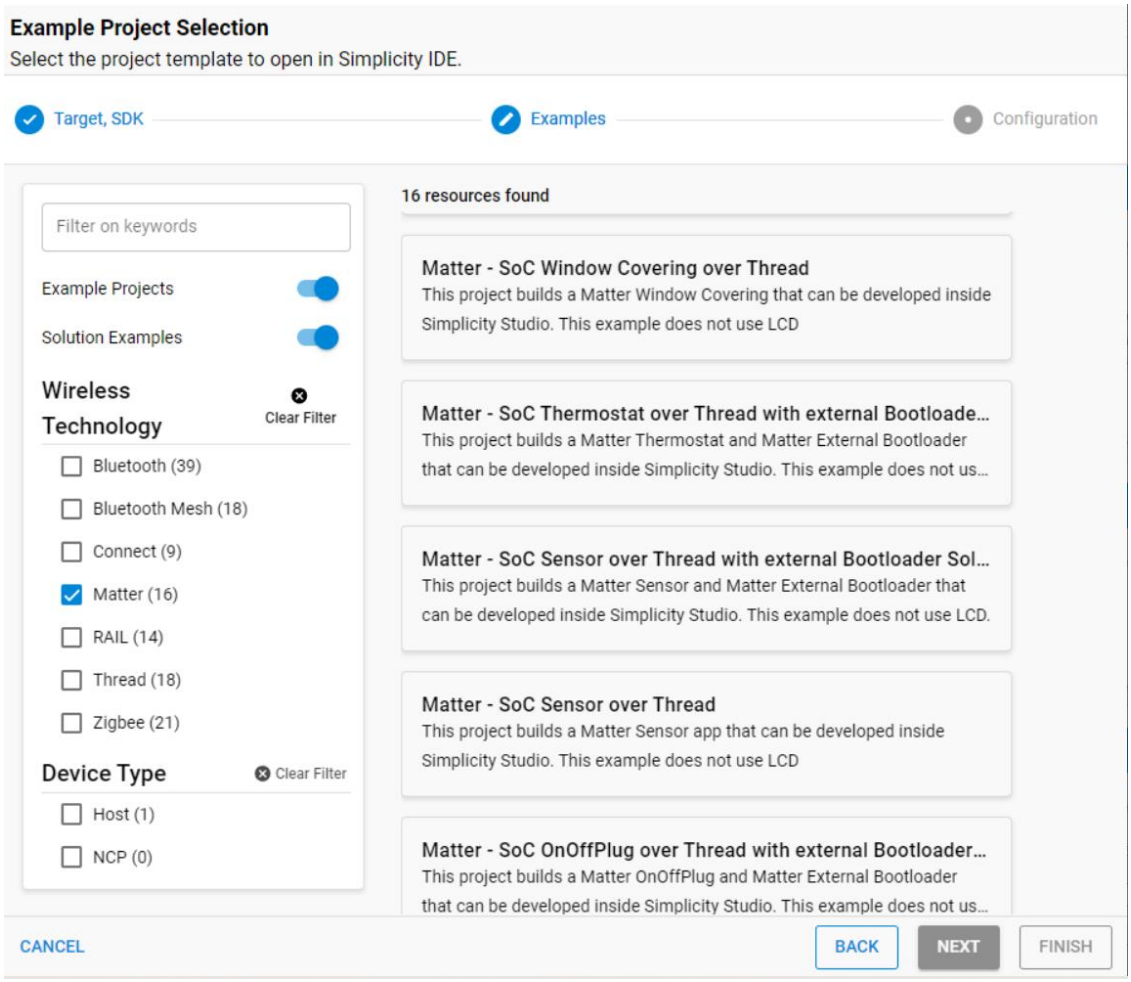


Figure 36 Gecko SDK - Matter examples

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

6.3. Zigbee Cluster Configurator

If a Matter example needs to be modified to add an endpoint, for example. This can be done using the Zigbee Cluster Configurator (ZCL), which can be found in the project file (.slcp) under configuration tools.

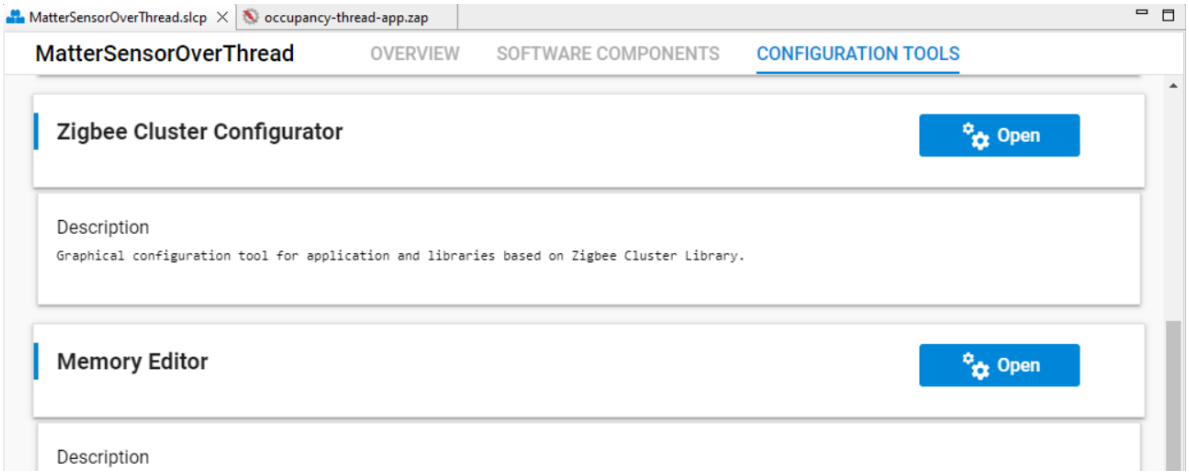


Figure 37 Zigbee Cluster Configurator

6.3.1. Adding endpoint

Once the Configurator is open, you can add a new endpoint. Click on 'Add Endpoint' and a window should appear. In this window you can give this endpoint an ID and select a device type under 'Device', e.g. 'Matter Smoke CO Alarm'. Then click 'Create' and the endpoint is added.

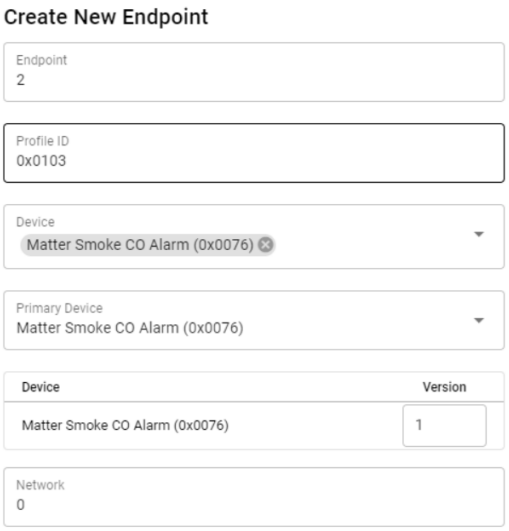


Figure 38 Adding Endpoint

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

6.3.2. Modifying endpoint

It's also possible to modify an endpoint and add or delete clusters. For example, we are creating a smoke sensor but we also want to measure the temperature (not a mandatory cluster), first check the application cluster specification so you know which cluster you need. Then we can add that cluster to this endpoint. To do this, click on the endpoint and on the right hand side you will see the same tabs as in the application cluster specification. Click on 'Measurement & Sensing', you will see 'Temperature Measurement', click on 'Enable' and select Server. We chose Server because we only want to read the temperature. This will add the cluster.

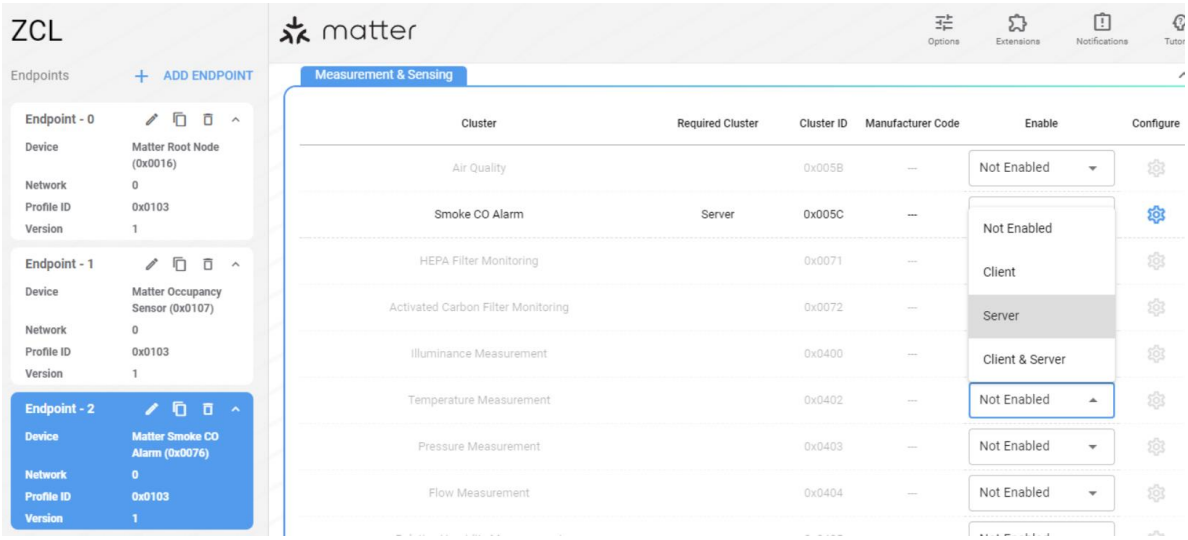


Figure 39 Modifying Endpoint

By clicking on the Configure button of the cluster, it is possible to enable or disable attributes. For example, the smoke sensor can only be used for 10 years because of the life cycle of the sensor. There is an attribute 'ExpiryDate' in the 'Smoke CO Alarm' cluster that we want to use to achieve this. Click on the 'Configure' button, locate the 'ExpiryDate' attribute and enable it. It is also possible to select the location.

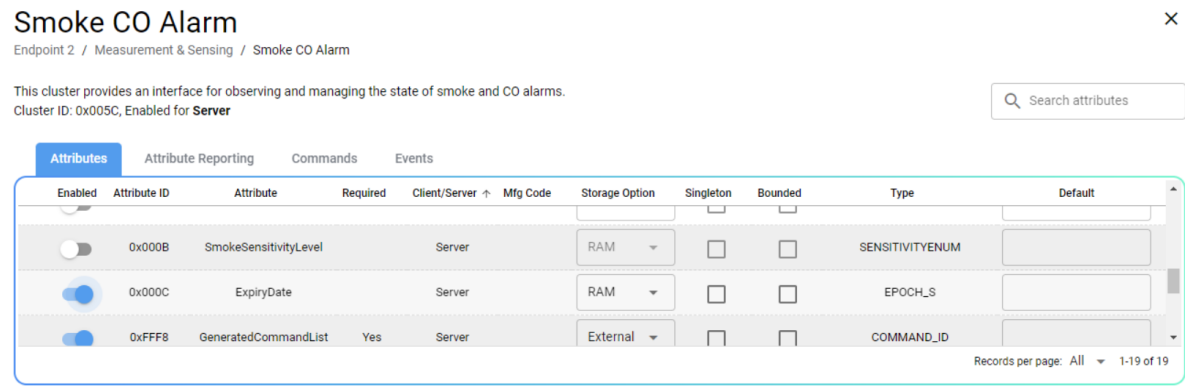


Figure 40 Modifying Attributes

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

This allows you to modify your endpoint as you wish. New clusters need to be installed in the software components, normally this is done automatically when using the ZCL.

6.3.3. Custom Cluster

If an Attribute and/or Command is not available in an existing cluster it is possible to add a custom cluster. There are several ways to do this but the best way is to use sample-extensions.xml located in user\SimplicityStudio\SDKs\gecko_sdk\app\zcl\sample-extensions.xml OR [gecko_sdk/app/zcl/sample-extensions.xml at 911f6cdefccbae03bc66e8c790ceb7e67ca07417 · SiliconLabs/gecko_sdk (github.com)](#) as a reference where everything is explained.

After writing a cluster configuration go to ZCL and then to extensions. A window should appear, you can browse your file and add it.

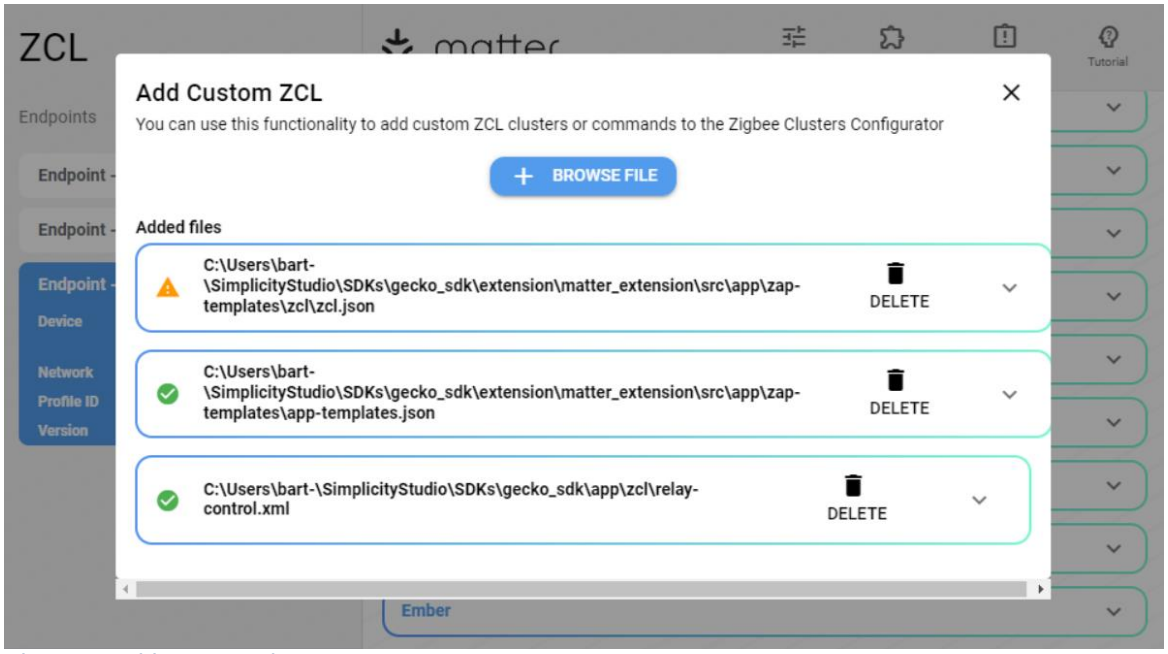


Figure 41 Add Custom Cluster

Once this is done, your cluster should appear. Activate it by server or client or both, save the file and the necessary changes will be made automatically.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

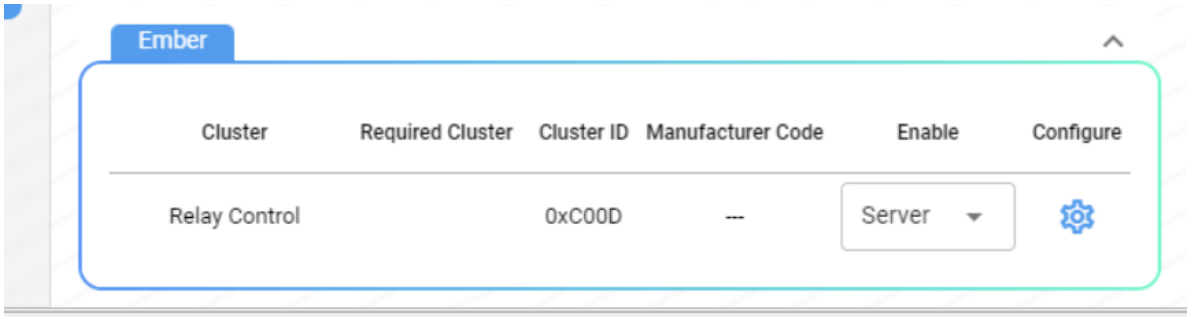


Figure 42 Enable Custom Cluster

6.4. Software Components

Projects are configured by installing and uninstalling components and configuring installed components. On the left you can see all the components in categories and on the right all the details about the selected component. For example, I want to measure the temperature of the cluster I have configured in the ZCL using the 'On-Board Sensor', which can be found under 'Board Drivers' on the left, and can be installed on the right.

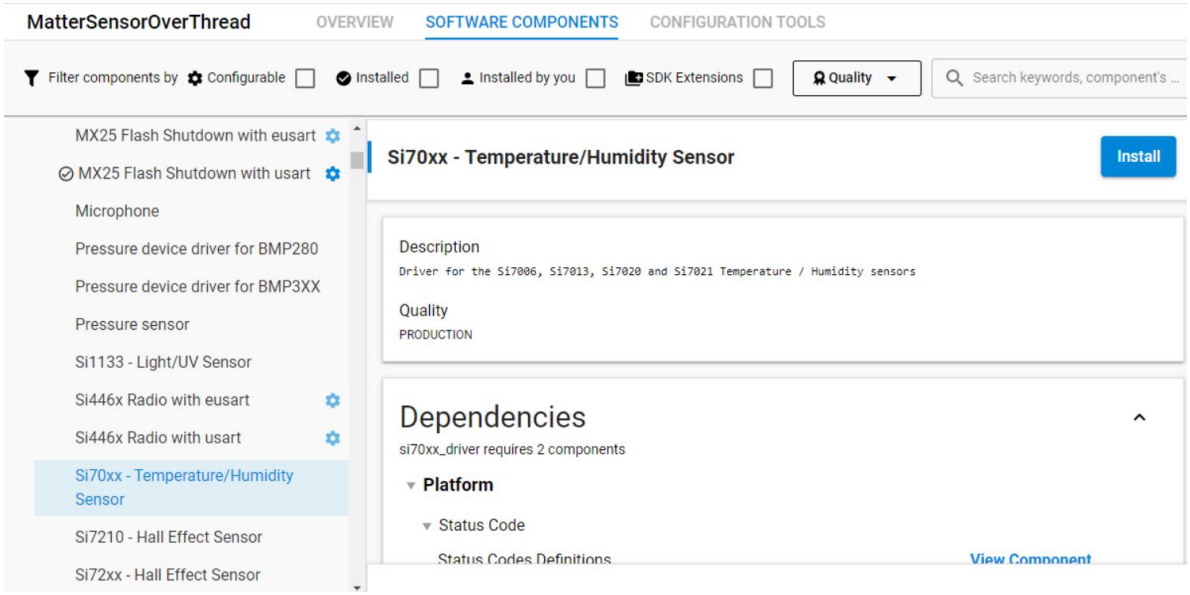


Figure 43 Software components

7. Setting up a Matter Hub

As mentioned in section 4, I used a Raspberry Pi as the Matter Hub in this project. For ease of use, I chose to use a pre-built Matter Hub developed by Silicon Labs, which consists of the Open Thread Border Router (OTBR) and the chip-tool running on a Raspberry Pi. The chip-tool is a Matter controller that allows easy development and testing using the command line.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

7.1. Silicon Labs Hub image

Silicon Labs have created a Raspberry Pi image that combines the OTBR and the chip tool, which can be downloaded and flashed onto an SD card, which is then inserted into the Raspberry Pi. See the Silicon Labs docs for full instructions and the link to the image. Docs and image can be found here https://docs.silabs.com/matter/latest/matter-thread/raspi-img

7.2. Radio Co-Processor

The Radio Co-Processor (RCP) is a Thread device that connects to the Raspberry Pi via USB. To flash the RCP, connect it to your laptop via USB. Thereafter, it should be connected to the Raspberry Pi via USB as well. Prebuilt RCP images developed by Silicon Labs and instructions are located here: https://docs.silabs.com/matter/latest/matter-thread/matter-rcp

7.3. Chip-tool

The chip-tool is a Matter controller implementation that allows to commission a Matter device into the network and to communicate with it using Matter messages, which may encode Data Model actions, such as cluster commands. Within the prebuilt Matter Hub image, silicon labs has simplified this.

Basic Mattertool Commands

Command	Usage
mattertool startThread	Starts the thread network on the OTBR
mattertool bleThread	Starts commissioning of a Matter Accessory Device using the chip-tool
mattertool on	Sends the *on* command to the Matter Accessory Device using the chip-tool
mattertool off	Sends the *off* command to the Matter Accessory Device using the chip-tool

You can also use the full chip-tool command set (still using mattertool):

```
$ mattertool levelcontrol read current-level 106 1
```

Figure 44 Mattertool commands

As they say, the full use of the chip-tool command set is still available, Just replace "./chip-tool" with "mattertool" because this is an alias and makes it easier. More information about commands is located here:
https://github.com/project-chip/connectedhomeip/blob/master/docs/guides/chip_tool_guide.md

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

8. Matter Tests

8.1. Simple on/off device

The first test was a simple on/off device to see what the protocol does.

The first step is to setup a Matter Hub. The seconds step was flashing the Radio Co-Processor (RCP) into the board. This was very easy with Simplicity Studio v5. Open SSv5, connect the board its auto detected. Then under “Example projects & demos” select “Thread” in Wireless Technology and then click on create. After the project is created, flash the project into the board. The third step was flashing the demo “SoC Lighting over Thread” into the board. The demo is located under “Example projects & demos” under the section Matter. The demo can be directly flashed into the board.

After setting up the Matter Hub and flashing the demo, we'll need to login to the Matter Hub and start testing the protocol using the chip-tool.

First we need is to start the Thread Network using **mattertool startThread** command. This creates a new Thread Network.

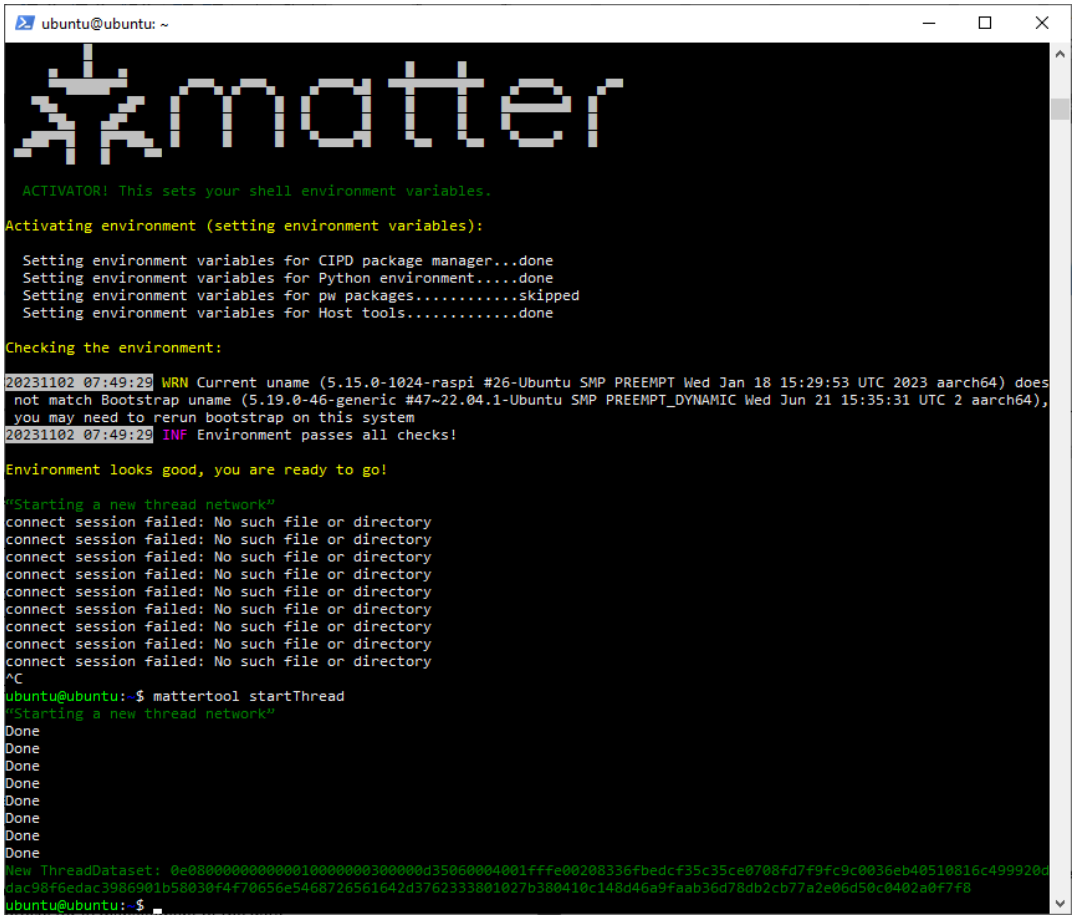


Figure 45 On/Off test - Output mattertool startThread

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

When the network is created, the device can be commissioned by using the command **mattertool bleThread**. When the device is successful commissioned it gives the Node-id.

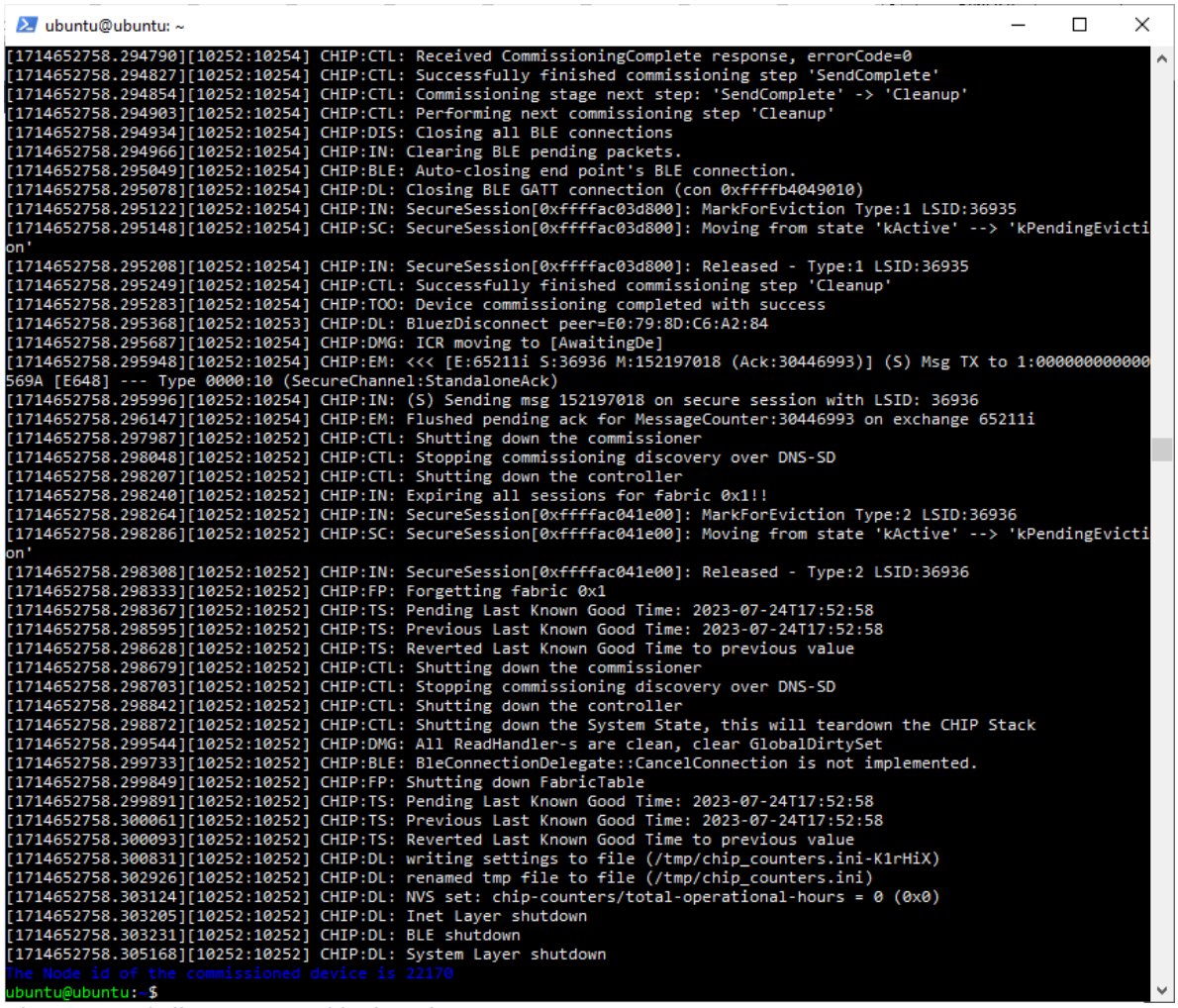


Figure 46 On/Off test - Output bleThread

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

After that we can control the light using the command **mattertool on** and **mattertool off**. With this command the devices changes there on/off state to on or off.

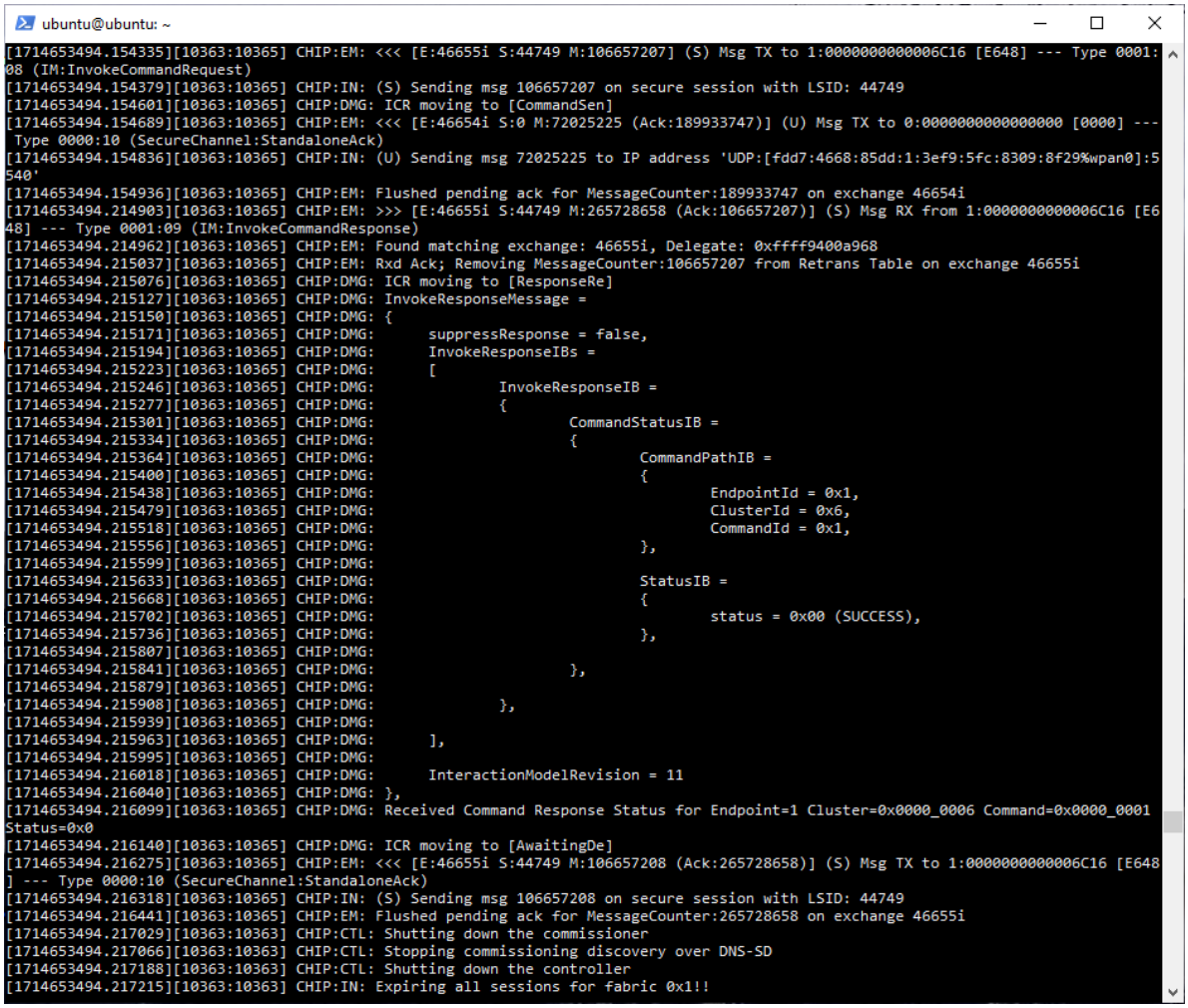


Figure 47 On/Off test - mattertool On/Off test

In this example we can see that the command **mattertool on** uses the invoke interaction which is explained in section 5.6.4. As we see in the cluster specification ClusterId 0x6 is the on/off cluster and the CommandId 0x1 is the on command.

So now I can control the light with these commands, but I want to know if I can monitor it. After a little research in the Matter docs under chip-tool guide, with the subscribe interaction in mind, I found how to do this. This guide describes how to subscribe to an attribute and this was exactly what I needed. When you subscribe to an attribute, any change will be shown on the command line. To subscribe we need to follow a few steps.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

1. Start the Chip-tool in interactive mode using **mattertool interactive start** after this command we can see “>>>” this means that we are in interactive mode. When the interactive mode is started it shows a list of all available clusters.

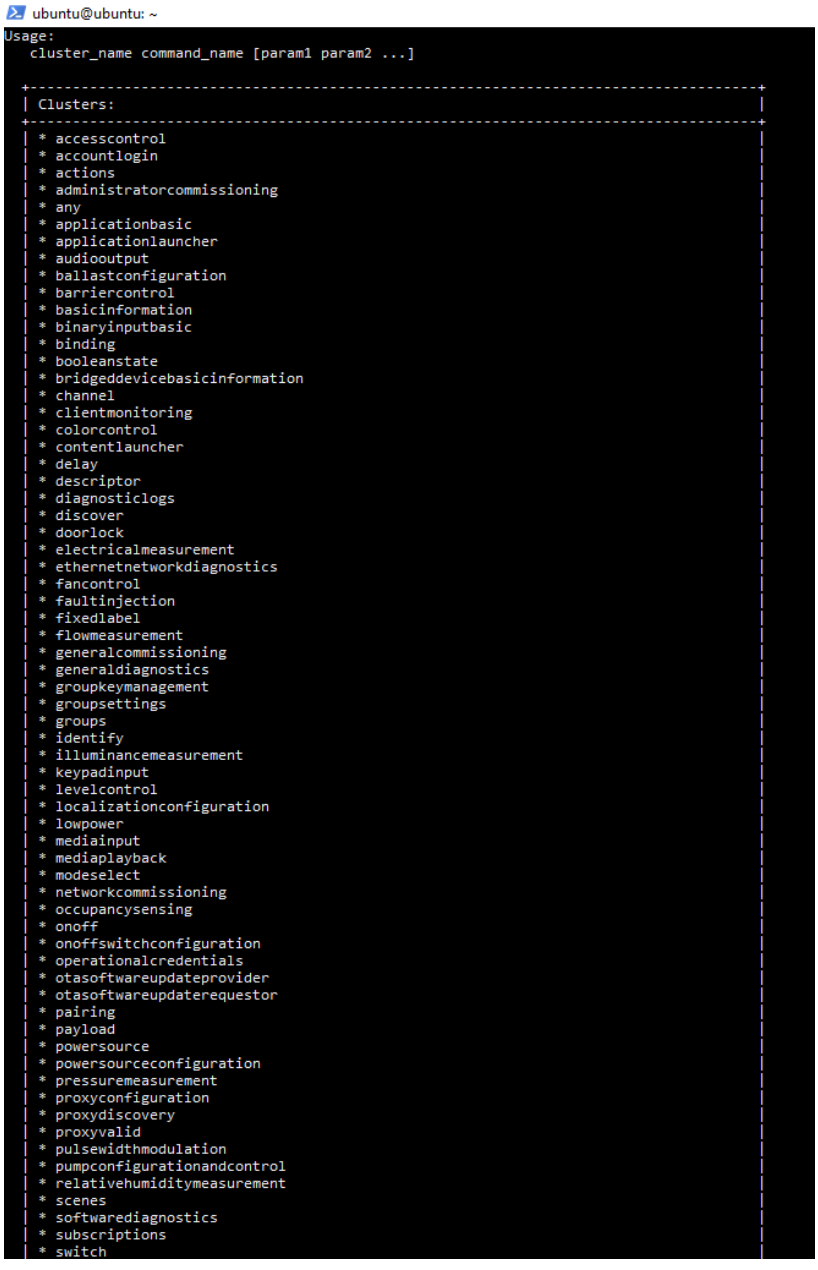


Figure 48 On/Off test - mattertool interactive

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

2. We want to monitor the on/off cluster. Use the command "**<cluster-name of choice> (onoff) subscribe**". The list of all available attributes for the cluster will appears.

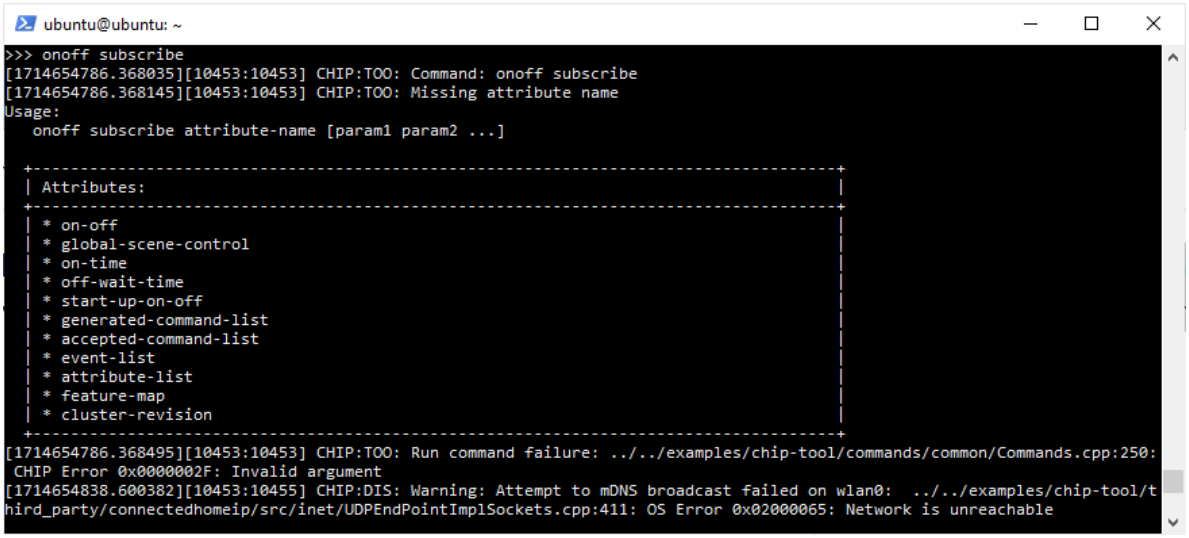


Figure 49 On/Off test - Subscribe On/Off available attributes

3. I need the On/Off attribute to monitor the changes. The base command that I used is:

   <cluster-name> subscribe <argument> <min-interval> <max-interval> <node_id> <endpoint_id>

```
>>> onoff subscribe on-off 1 1 27670 1
```

   The parameters of 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.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

<endpoint_id> is the ID of the endpoint where the chosen cluster is implemented.

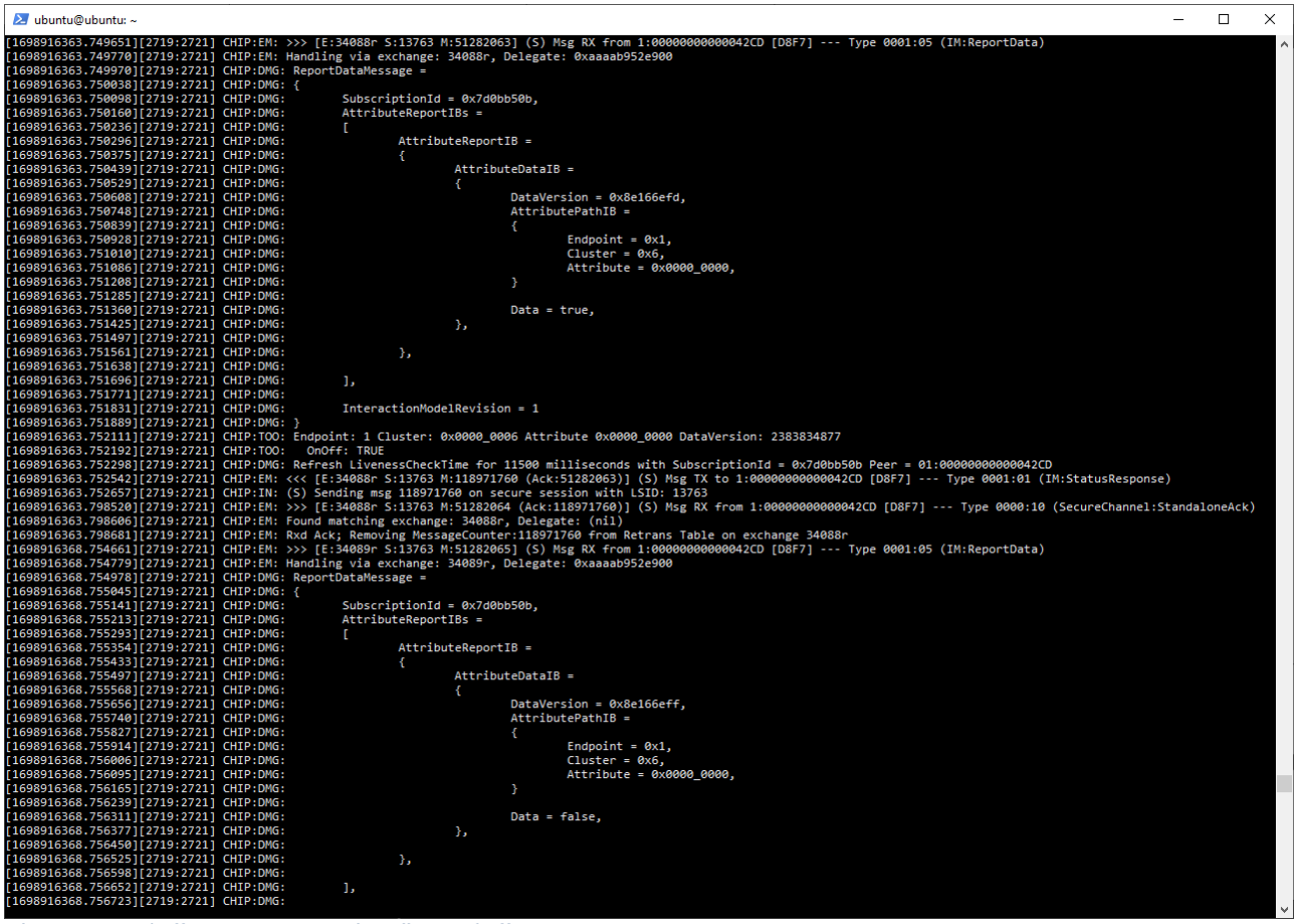


Figure 50 On/Off test - Output subscribe On/Off

So now every time I push the user button and the led changes, I get an update.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

8.2. Level control test

Before continuing this section please read 10.1 Accelerometer sensor problem

To test level control I started with the MatterSensorOverThread example and modify the endpoints in the Zigbee Cluster Configuration tool.

First step was the Zigbee Cluster Configuration tool in Simplicity Studio v5. I added a new endpoint with the device type “Matter On/Off Sensor”. I want to use this cluster to store the data in the future.

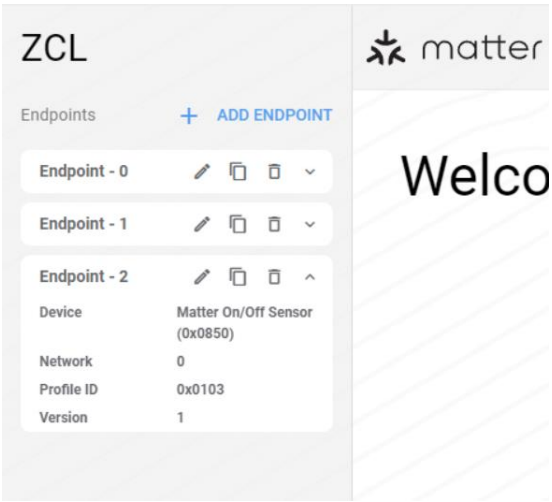


Figure 51 Level Control test - Add On/Off Server Endpoint

After that I enabled the level control cluster as a Client & Server in the General tab, actually we need only the Server cluster but I select both to test this.

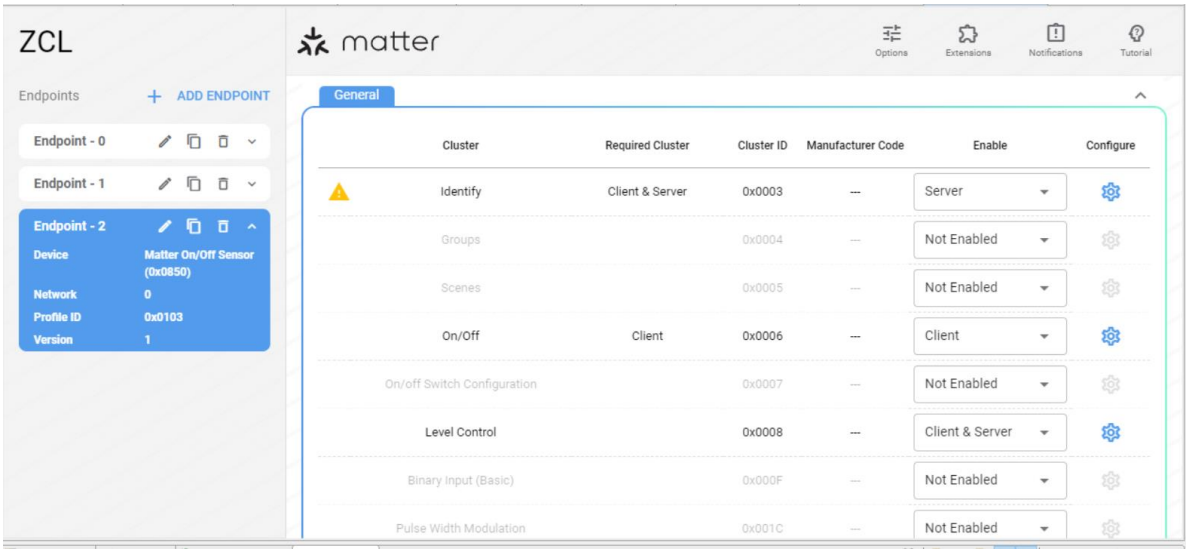


Figure 52 Level Control test – enabled Level Control Cluster

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

In the example project I found how they used a sensor en stored the data into an Attribute of a Cluster. In the SensorsCallback file we can see what they do when a button is pressed to activate the sensor.

```
68 void SilabsSensors::ActionTriggered(AppEvent * aEvent)
69 {
70 #if SL_SENSOR_TYPE == 1
71     if (aEvent->Type == AppEvent::kEventType_Button)
72     {
73         if(aEvent->ButtonEvent.Action == SL_SIMPLE_BUTTON_PRESSED)
74         {
75             mIsSensorTriggered = true;
76         }
77         else if (aEvent->ButtonEvent.Action == SL_SIMPLE_BUTTON_RELEASED)
78         {
79             mIsSensorTriggered = false;
80         }
81         UpdateBinarySensor(mIsSensorTriggered);
82         chip::DeviceLayer::PlatformMgr().LockChipStack();
83         halOccupancyStateChangedCallback(1, static_cast<HalOccupancyState>(mIsSensorTriggered));
84         chip::DeviceLayer::PlatformMgr().UnlockChipStack();
85     }
86
```

Figure 53 Level Control test - SilabsSensors::ActionTriggered

We can see that they call another callback function “halOccupancyStateChangedCallback”. This function is located in the “occupancy-server.h” file in the clusters folder.

```
72 void halOccupancyStateChangedCallback(EndpointId endpoint, HalOccupancyState occupancyState)
73 {
74     chip::BitMask<OccupancyBitmap> mappedOccupancyState;
75     if (occupancyState & HAL_OCCUPANCY_STATE_OCCUPIED)
76     {
77         mappedOccupancyState.Set(OccupancyBitmap::kOccupied);
78         emberAfOccupancySensingClusterPrintln("Occupancy detected");
79     }
80     else
81     {
82         emberAfOccupancySensingClusterPrintln("Occupancy no longer detected");
83     }
84
85     Attributes::Occupancy::Set(endpoint, occupancyState);
86 }
87
```

Figure 54 Level Control test - halOccupancyStatChangedCallback

On line 85 they set the state by using Attributes::Occupancy::Set(). This is what we need for our sensor. We are going to do the same for the level-control. The only thing that needs to be done is to set the current level attribute, which can be done with the following function.

chip::app::Clusters::LevelControl::Attributes::CurrentLevel::Set(endpoint, *val);

To use the set function to change the value of the level control attribute we have to add:

```
#include <app-common/zap-generated/attributes/Accessors.h>
```


Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

To test this properly, the intention is to create a virtual level that increments from 0 to 254 and back every x times. With this virtual level I can do the monitoring from the command line. To do this I had to make some changes to the AppTask.c file.

First I created a function “SetLevelVal” that sets the level of a specific endpoint.

```
/* ********** Set the value to the current-level attribute ********** */
void AppTask::SetLevelVal(EndpointId endpoint, uint8_t *val)
{

  unsigned char ValChar = *val;
  chip::DeviceLayer::PlatformMgr().LockChipStack();
  chip::app::Clusters::LevelControl::Attributes::CurrentLevel::Set(endpoint, ValChar);
  chip::DeviceLayer::PlatformMgr().UnlockChipStack();
}
```

Figure 55 Level Control test - SetLevelVal

Then I created a callback function that can be called every x times to set the value. This will increment or decrement the value from 0 to 254 and then call the SetLevelVal function.

```
/* ********** Timer callback ********** */
void AppTask::VirtualLevelTimerCallback(TimerHandle_t xTimer)
{

  if (lvlVal == 254)
    {
      incrementlvl = -1;
    }
  else if (lvlVal == 0)
    {
      incrementlvl = 1;
    }

  lvlVal += incrementlvl;

  SetLevelVal(LevelControlEndpoint, &lvlVal);

}
```

Figure 56 Level Control test - VirtualTimerCallback

To ensure that the callback function is called every x times, I had to create a function that initializes, sets the callback and starts the timer.

```
/* ********** Initialize and start timer ********** */
CHIP_ERROR AppTask::LevelControlTaskInit()
{

  VirtualLevelTimer = xTimerCreate("LvlTmr", pdMS_TO_TICKS(250), true, (void *) this, VirtualLevelTimerCallback);
  if (VirtualLevelTimer == NULL)
    {
      SILABS_LOG("Virtual level Timer create failed");
      appError(APP_ERROR_CREATE_TIMER_FAILED);
    }
  xTimerStart(VirtualLevelTimer,0);

  return CHIP_NO_ERROR;

}
```

Figure 57 Level Control test - LevelControlTaskInit

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

The only thing that needs to be done is to implement this in the main task.

```
184  void AppTask::AppTaskMain(void * pvParameter)
185  {
186      AppEvent event;
187      QueueHandle_t sAppEventQueue = *(static_cast<QueueHandle_t *>(pvParameter));
188
189      CHIP_ERROR err = sAppTask.Init();
190      if (err != CHIP_NO_ERROR)
191      {
192          SILABS_LOG("AppTask.Init() failed");
193          appError(err);
194      }
195
196      /* ********** Initialize the level control timer to test level control ********** */
197      err = LevelTest.LevelControlTaskInit();
198      if (err != CHIP_NO_ERROR)
199      {
200          SILABS_LOG("LevelControlTastInit() failed");
201          appError(err);
202      }
203      //
204
205  #if !(defined(CHIP_DEVICE_CONFIG_ENABLE_SED) && CHIP_DEVICE_CONFIG_ENABLE_SED)
206      sAppTask.StartStatusLEDTimer();
207  #endif
208
209      SILABS_LOG("App Task started");
210
211      while (true)
212      {
213          BaseType_t eventReceived = xQueueReceive(sAppEventQueue, &event, portMAX_DELAY);
214          while (eventReceived == pdTRUE)
215          {
216              sAppTask.DispatchEvent(&event);
217              eventReceived = xQueueReceive(sAppEventQueue, &event, 0);
218          }
219      }
220  }
221
```

Figure 58 Level Control test - AppTaskMain

8.2.1. Capture data using chip-tool

To capture the data we can use the interactive function in the chip-tool and subcribe to the endpoint. Every time the data changes it will be displayed. The following commands is being used:

1. Mattertool interfactive start
2. Levelcontrol subscribe current-level (min-interval: I used 0) (max-interval: I used 1) (nodeId) (endpointId)

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

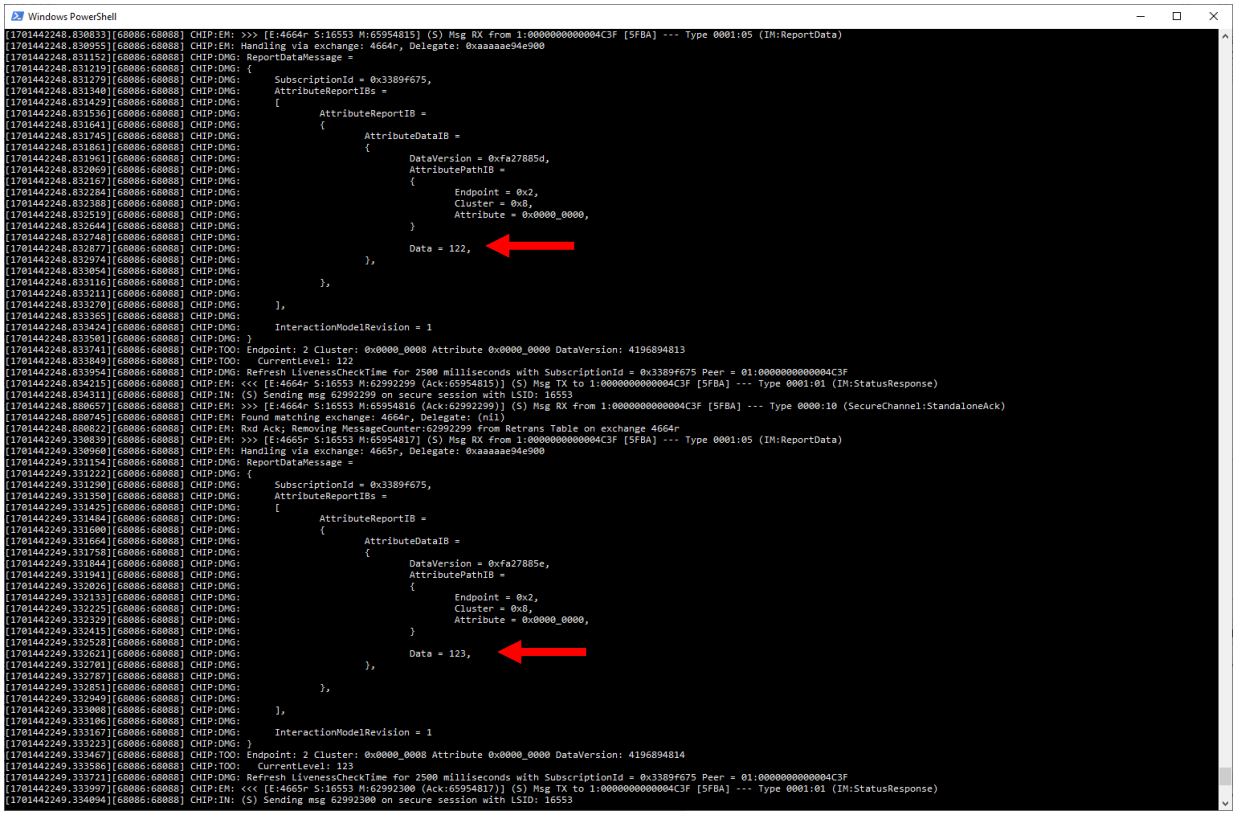


Figure 59 Level Control test - Capture data

With this output I can see that the level control is working.

8.3. Terminal output test

Finally, we want to use the terminal output to control the game. In the previous tests, the data is successfully displayed in the terminal, but it shows a lot of information I don't need. I tried to use grep, but because the chip-tool interactive mode is a process, this was very difficult.

I had to make sure that a script would do this for me, so that I could be sure that the right command was executed at the right time. I used Python and the subprocess module to do this. I created a test.py file, and in this script we will use interactive mode, just like we did manually at first. We do this by defining the commands and starting a process with Popen.

```
6
7   nodeId = input("Enter node id:")
8   #nodeId = "8879"
9
10  command = "/home/ubuntu/scripts/matterTool.sh interactive start"
11  levelcontrol = "levelcontrol subscribe current-level 0 1 " + nodeId + " 2\n"
12
13  proc = Popen([command], shell=True, text=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)
```

Figure 60 Terminal Output test - Subproces Popen

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Now we need to know when we can send the command to subscribe to the level-control current-level attribute. To do this, we need to parse the startup and see what the last line is so that we can use that as an indication that the startup is complete.

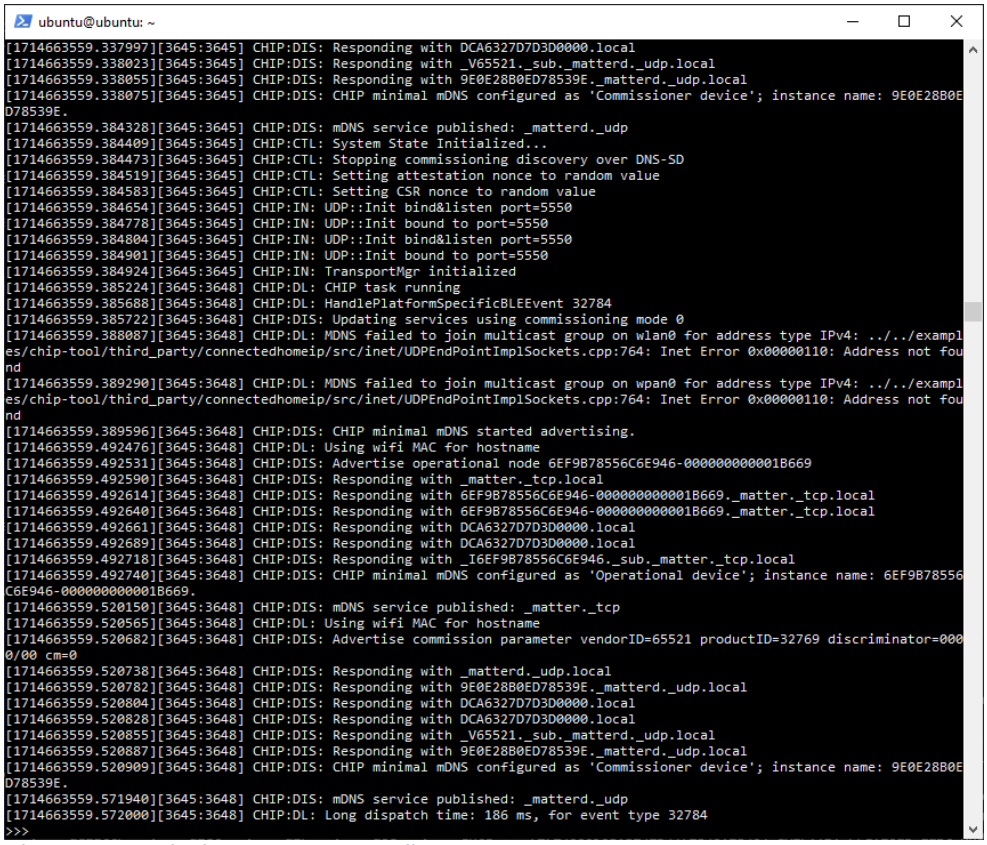


Figure 61 Terminal Output test - Last line

This way I know the last line, and all I have to do is use it as a reference. So I will look for the string 'Long dispatch'. When this string is found, we can use the following command to subscribe to the attribute.

```
13  proc = Popen([command], shell=True, text=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)
14
15  line_found = False
16  while not line_found:
17      line = proc.stdout.readline()
18      print(line)
19      if line.find("Long dispatch") > -1:
20          line_found=True
21
22  print("We reached the end of the startup")
23  proc.stdout.flush()
24  proc.stderr.flush()
25  time.sleep(5)
26  print(dir(proc.stdin))
27  proc.stdin.write(levelcontrol)
28  proc.stdin.close()
```
Figure 62 Terminal Output test - Search last line

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Now it's important to read each line. This way we can filter out only the data we need.

```
30  line_found = False
31  while not line_found:
32
33      line = proc.stdout.readline()
34
35      try:
36          data_index = line.index("Data =")
37      except ValueError:
38          data_index = False
39
40
41      if data_index != False:
42          datastream = line.split("Data = ")[1].split(",")[0]
43          print(datastream)
44
```

Figure 63 Terminal Output test - Filter data

This will ensure that if a line containing 'Data =' is found in the process, it will only be filtered to the number that comes after '=' and before ','. This is the number we will ultimately need to control the game.

```
[1701442249.332133][68086:68088] CHIP:DMG:                                  Endpoint = 0x2,
[1701442249.332225][68086:68088] CHIP:DMG:                                  Cluster = 0x8,
[1701442249.332329][68086:68088] CHIP:DMG:                                  Attribute = 0x0000_0000,
[1701442249.332415][68086:68088] CHIP:DMG:                              }
[1701442249.332528][68086:68088] CHIP:DMG:
[1701442249.332621][68086:68088] CHIP:DMG:                              Data = 123,
[1701442249.332701][68086:68088] CHIP:DMG:                          },
[1701442249.332787][68086:68088] CHIP:DMG:
[1701442249.332851][68086:68088] CHIP:DMG:                      },
[1701442249.332949][68086:68088] CHIP:DMG:
[1701442249.333008][68086:68088] CHIP:DMG:              ],
```

Figure 64 Terminal Output test - Data example

So, what this script will do is first ask for the node ID that we received during commissioning process. Then the script will start the process, wait until it's up and running, and then subscribe to the current-level attribute. It will then filter it so that only the usable value is left, store it in a variable, and then print it. This test can then be used later in the game itself.

```
[1714660995.635882][3083:3086] CHIP:DL: Long dispatch time: 192 ms, for event type 32784

We reached the end of the startup
['_CHUNK_SIZE', '__class__', '__del__', '__delattr__', '__dict__', '__dir__', '__doc__', '__enter__', '__eq__', '__exi
t__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__iter__', '
__le__', '__lt__', '__ne__', '__new__', '__next__', '__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeo
f__', '__str__', '__subclasshook__', '_checkClosed', '_checkReadable', '_checkSeekable', '_checkWritable', '_finalizin
g', 'buffer', 'close', 'closed', 'detach', 'encoding', 'errors', 'fileno', 'flush', 'isatty', 'line_buffering', 'name'
, 'newlines', 'read', 'readable', 'readline', 'readlines', 'reconfigure', 'seek', 'seekable', 'tell', 'truncate', 'wri
table', 'write', 'write_through', 'writelines']
106
107
106
107
107
107
106
106
```

Figure 65 Terminal Output test - Output filtered data

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

9. ICM-20689 Inertial sensor

The ICM-20689 is a 6-axis inertial sensor that combines a 3-axis accelerometer and a 3-axis gyroscope. This sensor is integrated on the development board. There is a software component available in the Gecko SDK that contains the driver for this sensor. The sensor datasheet contains all the information and registers needed to configure and use the sensor.

For this project, both the gyroscope and the accelerometer can be used, but there are two differences. The gyroscope measures speed (°/s), so when the sensor is moving, we get a value of how fast the sensor is moving. When the sensor stops moving, we get a "0" value.
The accelerometer measures gravity (g). The sensor keeps the value at its position even if the sensor stops moving. It is easier to use the accelerometer because we can assign the position to a specific value.

9.1. Using the ICM-20689 driver

The driver contains all functions necessary to use the sensor. Silicon Labs has a driver for this sensor that contains all the functions needed to use the sensor. The driver can be used by installing the software component in simplicity studio. Open the project file (.slcp) and select Software Components. Look for ICM20689 and install the component.

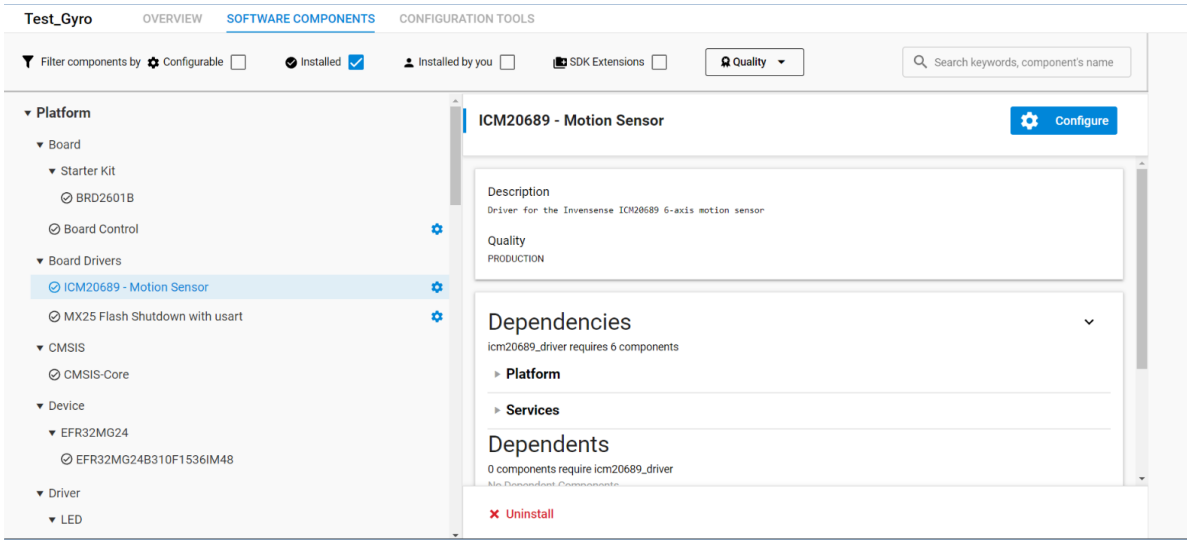


Figure 66 ICM-20689 - Driver Software components

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

After installed the driver set PC09 too high to enable the sensor, refer to section 2.3. There are two options to do this, by using the pintool or manually.

- By using the pintool:

  Select PC09, then click on edit

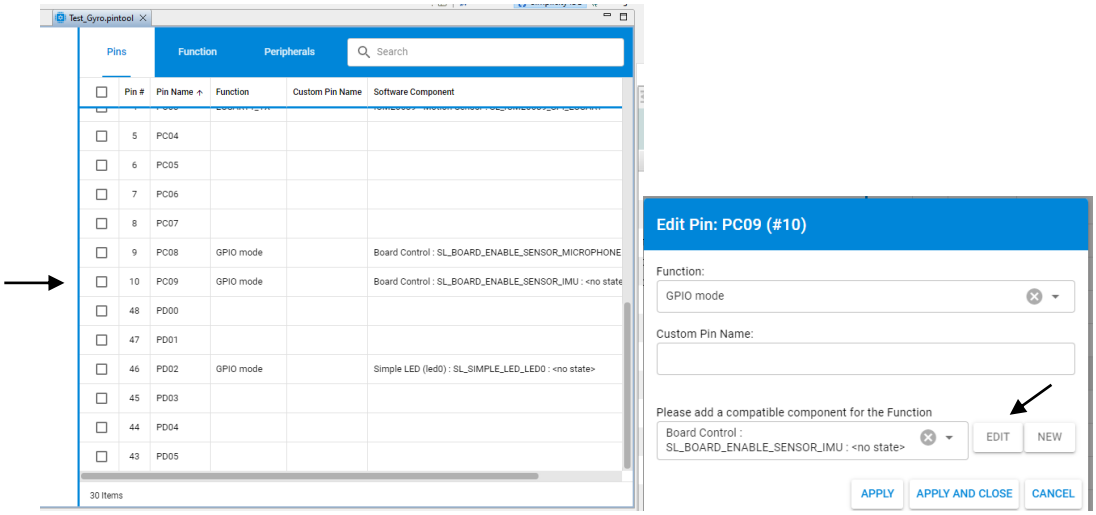


Figure 67 ICM-20689 - Enable sensor pin

After that click on Enable Inertial Measurement Unit.

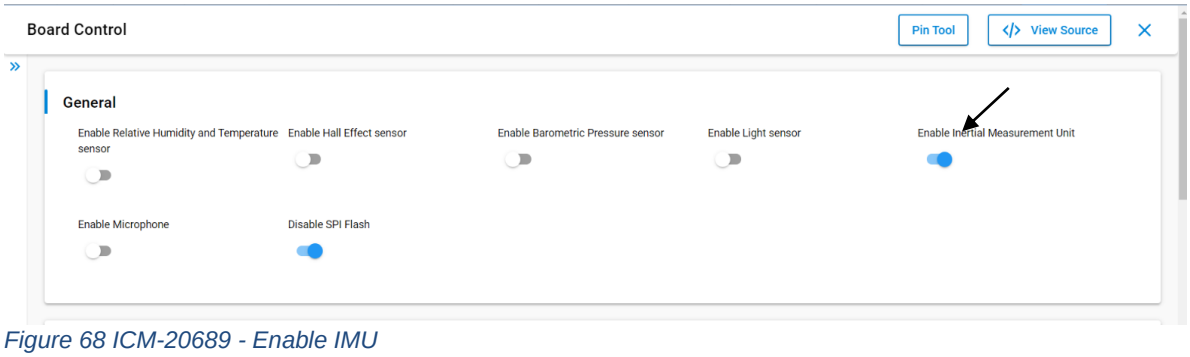


Figure 68 ICM-20689 - Enable IMU

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

- Manually:
  To enable the sensor go to the folder config in the project and open sl_board_control_config.h and set the define SL_BOARD_SENSOR_IMU to 1.

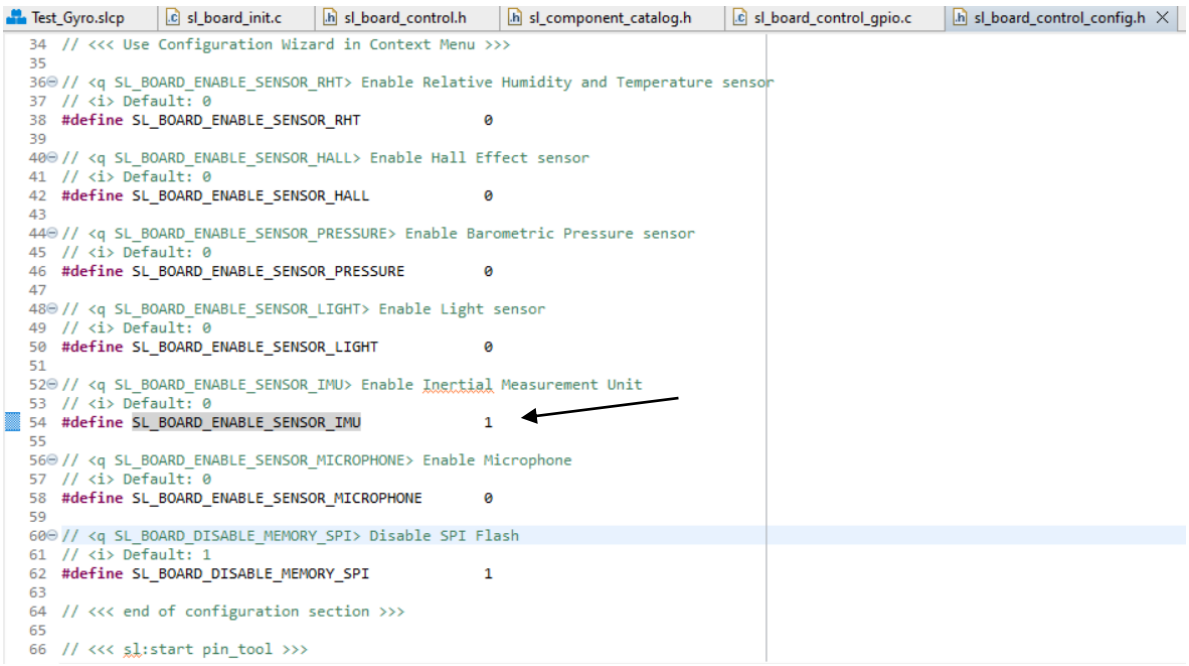


Figure 69 ICM-20689 - Enable sensor manually

9.2. Tests

9.2.1. Raw data

To test the motion sensor I have used a simple bare-metal project using the Gecko SDK. I added UART to send the sensor data to a terminal. To use UART I had to install the software component for this. I also created a tick timer to send the data with an interval of x ms.I created a file called message.c with some useful functions.

```
void tick_init(void);
void tick_cb(sl_sleeptimer_timer_handle_t *handle, void *data);
void UART_SendMessageLN(uint8_t *Message);
void UART_SendMessage(uint8_t *Message);
void UART_SendGyroData(float data[3]);
void UART_SendAccelData(float data[3]);
void callback(UARTDRV_Handle_t handle, Ecode_t transferStatus, uint8_t *data, UARTDRV_Count_t transferCount);
```

Figure 70 ICM-20689 test - Util functions

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Tick_init starts a periodic timer and the tick_cb is called each time the timer period expires. UART_SendMessageLN sends a string with linefeed at the end while UART_SendMessage does not. The UART_SendGyroData and UART_SendAccelData functions send the data coming from the sensor. This function expects an array so we can send the 3 axes at once and also uses the SendMessage functions to send the data. It also uses sprintf to convert the float value to a character.

```
67 void UART_SendAccelData(float data[3])
68 {
69   uint8_t converted[15];
70   UART_SendMessageLN((uint8_t*)"Accelerometer:\n");
71
72   UART_SendMessage((uint8_t*)"Y: ");
73   sprintf(converted, "%f", data[0]);
74   UART_SendMessageLN(converted);
75
76   UART_SendMessage((uint8_t*)"X: ");
77   sprintf(converted, "%f", data[1]);
78   UART_SendMessageLN(converted);
79
80   UART_SendMessage((uint8_t*)"Z: ");
81   sprintf(converted, "%f", data[2]);
82   UART_SendMessageLN(converted);
83
84   UART_SendMessageLN((uint8_t*)"\n ----------------------------");
85 }
```

Figure 71 ICM-20689 test - UART_SendAccelData

In the application function "app_process_action" the tick timer increments the variable g_tick every 1ms and every 250 milliseconds this block of code is executed. So every time this is executed, we read the sensor and store the data in a float array that is sent with the UART_SendAccelData function.

```
55 void app_process_action(void)
56 {
57   blink_process_action();
58   if(g_tick >= 250)
59     {
60       status = sl_icm20689_accel_read_data(data);
61       if (status == SL_STATUS_OK)
62         {
63           UART_SendAccelData(data);
64         }
65       else
66         {
67           UART_SendMessageLN((uint8_t*)"Something went wrong reading data");
68         }
69       g_tick = 0;
70     }
71
72 }
```

Figure 72 ICM-20689 test - app_process_action

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

When I run this code and tilt the board first to left and then right we get this data.

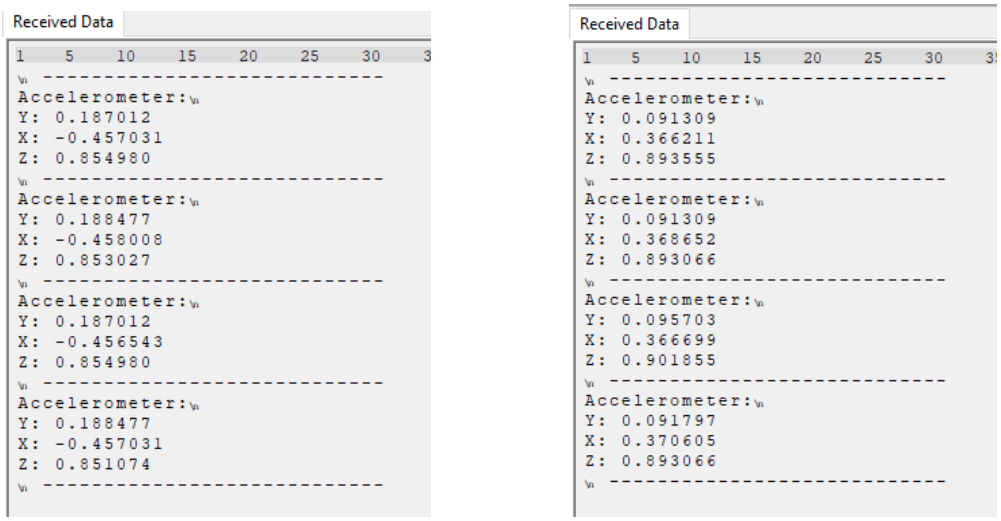


Figure 73 ICM-20689 test - Output raw data

So the most interesting value to read is the X-axis. We can see that when the board is tilted to the left we get a negative value and to the right we get a positive value. So that is what we are going to use.

9.2.2. Converted data

The floating point value returned by the sensor is not usable in the Matter application. In the Matter application, I use the Level Control attribute, which stores a uint8 data type. So we need to convert the floating point variable to a uint8 variable.

To do this, I created a function that does the conversion. In this function, we have defined the minimum and maximum limits for the input value, so that it does not exceed these values. Then the conversion from float to uint8 is done.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

```
uint8_t ConvertAccelFloat(float val)
{
  float minValue = -0.9;
  float maxValue = 0.9;

  if(val > maxValue)
    {
      val = maxValue;
    }

  else if(val < minValue)
  {
    val = minValue;
  }

  uint8_t uint8Value = (uint8_t*)((val - minValue) / (maxValue - minValue) * 254);

  return uint8Value;

}
```

Figure 74 ICM-20689 test - ConvertAccelFloat

10. Issues

In this section I'm going to tell all about the problems I faced during the research and the possible solutions I found.

10.1. Accelerometer sensor problem

When doing the research of the Matter protocol I was facing a big potential problem for the project. The protocol has pre-build "device type's" these are officially defined and they are not customizable. The only thing we can adjust is a custom cluster. A cluster contains a set of functionalities. This looks like a major problem because we want to read a sensor that is not included into the officially device type's and we DON'T want to manipulate the protocol.

10.1.1. Possible solution

- Digging deeper into the Device Library, I noticed that there's actually one device type we can use, On/Off Sensor. The On/Off sensor includes an optional cluster Level control. Level control can be anything and is not something specific.

1.6. Level Control

This cluster provides an interface for controlling a characteristic of a device that can be set to a level, for example the brightness of a light, the degree of closure of a door, or the power output of a heater.

This can solve the problem on the side of server so we can manipulate the current-level attribute at the server side, our sensor.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

- Another solution that I found was to create a custom app, combine this with the device type and the Level control cluster.

10.1.2. Solution

There is no good solution, I want to do something unintended with the protocol. First we had to decide whether to continue with this project or not. We decided to continue because we want to know what the protocol is capable of. Then I decided to use a device type that best fit my needs and add the level control to it. The On/Off sensor fits best and the level control cluster is optional so I decided to use this.

10.2. Connection between matter device (using chip-tool) and the “game”

At first I thought that the interaction between the matter protocol and the game could be done by the read commands or by subscribing to an attribute of an endpoint (for example the on/off attribute that holds the on/off state) using the chip tool. While doing some tests it seems that we get a lot of information and not just the state of a device, for example a ‘true’ value when the button is pressed. So getting those values from the chip-tool in the command line isn't that easy. I don't even know if it's possible to filter everything so that I only get the information I want. To control the game, I just need the value of the attribute, whether it is two on/off devices, one for left, one for right, or a level control.

10.2.1. Possible solution

Maybe to make this work I can build another device that can be controlled by the game controller and then use gpio's (with or without pwm) that can be connected to the raspberry pi and then read those pins so we can control the breakout game with those pins?

10.2.2. Solution

In the end, I decided to go with the level-control cluster after all. There's a possibility to write a Python script using the subprocess module to read the data line by line and then filter out the data we need. After that, this script should be implemented into the game.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

11. Firmware game controller

The purpose of the firmware is to detect in which direction we are moving the board. This information will eventually be used in the Breakout game so that we can control the paddle by moving the board.

As previously mentioned in section 10.1, we won't be using the protocol exactly as prescribed. Just like we did in test section 8.2, in this firmware, we'll also utilize a Silicon Labs example and modify it. We'll add a new endpoint with device type On/Off Sensor. Here, we'll add the level control cluster, which is optional for this device type. However, we won't use it as a Client but rather as a Server, which means we won't be using the protocol as intended.

7.8.4. Cluster Requirements

Each endpoint supporting this device type SHALL include these clusters based on the conformance defined below.

Table 19. On/Off Sensor Cluster Requirements

ID	Cluster	Client/Server	Quality	Conformance
0x0003	Identify	Server		M
0x0003	Identify	Client		M
0x0004	Groups	Client		O
0x0005	Scenes	Client		P, O
0x0006	On/Off	Client		M
0x0008	Level Control	Client		O

Figure 75 Firmware controller - On/Off sensor Cluster Requirements

For this project, we will be using Simplicity Studio v5 and the Gecko SDK Suite v4.4.0 with the Matter extension.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

11.1. Matter Solution

For the controller, I'm working with a solution that includes both the firmware as well as a bootloader.

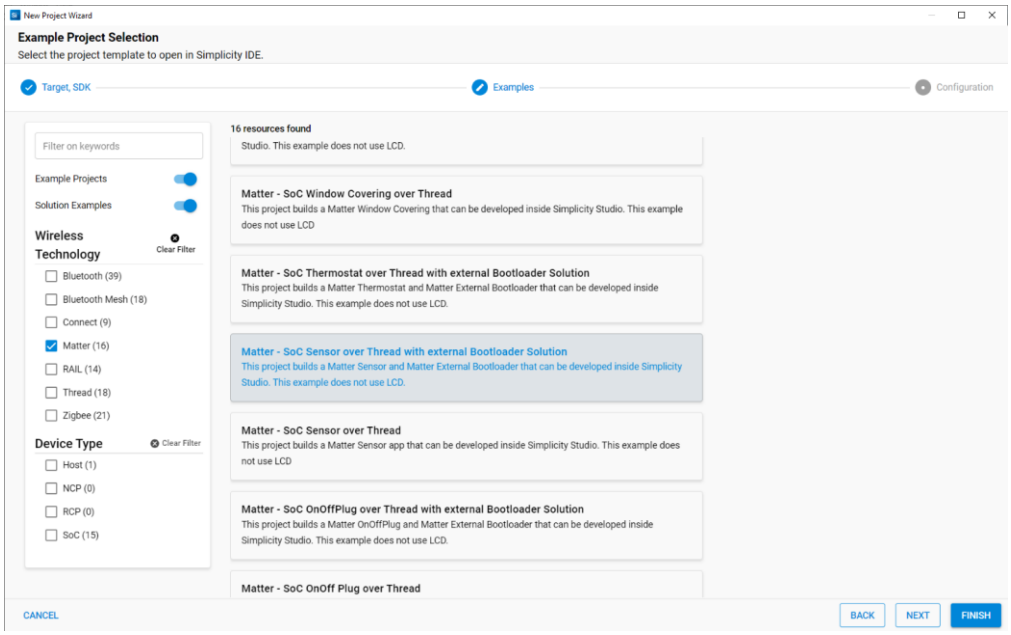
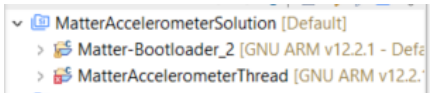


Figure 76 Firmware controller - Example solution

I only changed the name of the firmware project to ‘MatterAccelerometerThread’.



To test if everything works correctly, both the bootloader and the example application are build. Now they need to be uploaded to the SoC, and it's important to upload the bootloader first, followed by the application itself.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

11.2. Configuration

In the 'MatterAccelerometerThread' project, several adjustments are needed. We need to add the 'On/Off Sensor' endpoint and include the level control cluster server version there.

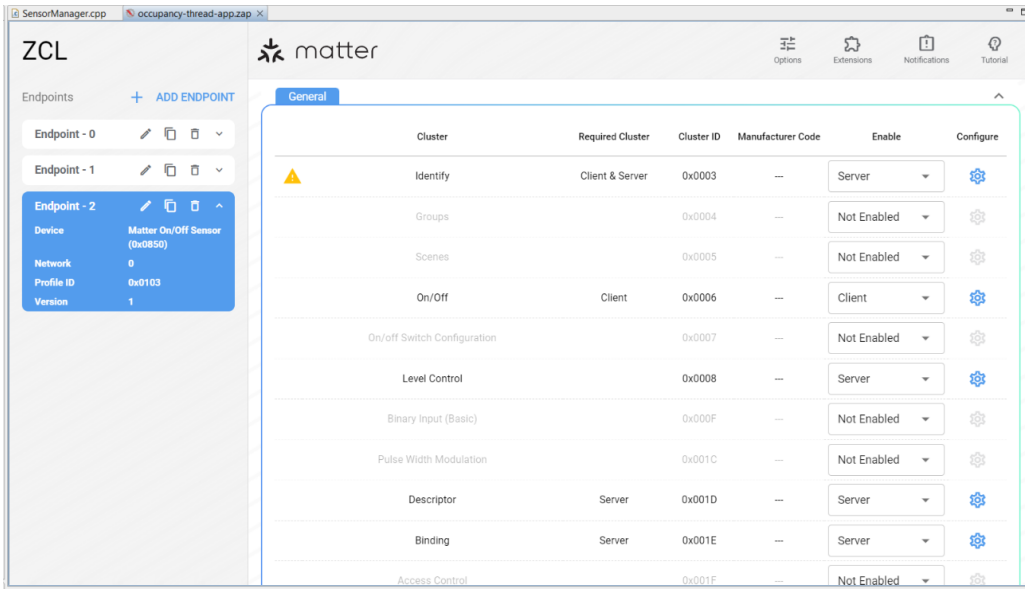


Figure 77 Firmware controller - Endpoint configuration

After the endpoint and cluster are configured, the sensor must be configured. This is exactly the same as in section 9.1..The only difference is that the level control cluster is set up as a server only.

11.3. SensorManager

To bundle all the functions and variables related to the sensor, I created a new file called 'SensorManager.cpp'. For all these functions I created a class called SilabsSensors.

```
class SilabsSensors
{
public:
    void InitSensor(void);

    static void ActionTriggered(AppEvent * aEvent);
    static void AcceleroTimerEventHandler(TimerHandle_t xTimer);
    uint8_t ConvertAccelFloat(float FloatVal);
    uint8_t ConvertToUint(float FloatVal);

private :
    static void SetLevel(chip::EndpointId endpoint, uint8_t *val);
    static bool mIsSensorTriggered;
    static void UpdateSensorActive(bool state);

};
```

Figure 78 Firmware controller - Class SilabsSensors

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

In this class there are several util functions. First, I reused the "SetLevel" function from section 8.2.. As explained, this sets the Current-Level attribute.

```
104 void SilabsSensors::SetLevel(EndpointId endpoint, uint8_t *val)
105 {
106   unsigned char ValChar = *val;
107
108   chip::DeviceLayer::PlatformMgr().LockChipStack();
109   chip::app::Clusters::LevelControl::Attributes::CurrentLevel::Set(endpoint, ValChar);
110   chip::DeviceLayer::PlatformMgr().UnlockChipStack();
111 }
```

Figure 79 Firmware controller - SetLevel

Another util function is “ConvertAccelFloat”. This function does the conversion from float to uint8 as explained earlier in section 9.2.2.

```
113 uint8_t SilabsSensors::ConvertAccelFloat(float FloatVal)
114 {
115   float minValue = -0.9;
116   float maxValue = 0.9;
117   uint8_t uintValue;
118
119   if(FloatVal > maxValue)
120     {
121       FloatVal = maxValue;
122     }
123
124   else if(FloatVal < minValue)
125   {
126       FloatVal = minValue;
127   }
128
129   uintValue = static_cast<uint8_t>((FloatVal - minValue)/(maxValue - minValue)*254);
130   return uintValue;
131 }
```

Figure 80 Firmware controller - ConvertAccelFloat

The UpdateSensorActive function will simply set the LED to the state we want. This is used to indicate that the sensor is reading or not.

```
148 void SilabsSensors::UpdateSensorActive(bool state)
149 {
150         AppTask::GetAppTask().SetAppLED(state);
151 }
152
```

Figure 81 Firmware controller - UpdateSensorActive

Then there's the ActionTriggered function. This is a callback function that is called from the moment the user button (btn1) is triggered. This function will set the game state to true or false depending on the previous state. The LED will also react accordingly. This only happens when the button is pressed and not released.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

```
void SilabsSensors::ActionTriggered(AppEvent * aEvent)
{
    if (aEvent->Type == AppEvent::kEventType_Button)
    {
        if(aEvent->ButtonEvent.Action == static_cast<uint8_t>(SilabsPlatform::ButtonAction::ButtonPressed))
        {
            if(game_state == GAME_PAUZE)
              {
                mIsSensorTriggered = true;
                game_state = GAME_RUNNING;
              }
            else if(game_state == GAME_RUNNING)
              {
                mIsSensorTriggered = false;
                game_state = GAME_PAUZE;
              }
        }
        UpdateSensorActive(mIsSensorTriggered);
    }
}
```

Figure 82 Firmware controller - ActionTriggered

"AcceleroTimerEventHandler" is the timer callback function that is called every x milliseconds. This function will read the sensor data and store it into a float array. Then it will convert only the x-axes, after that it will write the converted data into the attribute.

```
void SilabsSensors::AcceleroTimerEventHandler(TimerHandle_t xTimer)
{
  if (game_state == GAME_RUNNING)
    {
      uint8_t converted_data;
      status = sl_icm20689_accel_read_data(icm_data);
      if (status == SL_STATUS_OK)
        {
          converted_data = Sensor.ConvertAccelFloat(icm_data[1]);
          SetLevel(LevelControlEndpoint, &converted_data);
        }
    }
}
```

Figure 83 Firmware controller - AcceleroTimerEventHandler

The InitSensor function initializes the sensor and sets the scale. Then a timer is created that calls the callback function "AcceleroTimerEventHandler" every 30ms. After creating the timer, this function will start the timer after 1s to make sure everything is ok.

```
void SilabsSensors::InitSensor(void)
{
    chip::DeviceLayer::PlatformMgr().LockChipStack();
    status = sl_icm20689_init();
    if (status != SL_STATUS_OK)
    {
      SILABS_LOG("Accelero sensor initialization failed");
    }

    sl_icm20689_accel_set_full_scale(scale);
    chip::DeviceLayer::PlatformMgr().UnlockChipStack();

    AcceleroTimer = xTimerCreate("AcceleroTmr",              //Text name, not used by the RTOS kernel
                                 pdMS_TO_TICKS(30),          // timer period
                                 true,                       // reload timer
                                 (void *) this,              // Timer Id
                                 AcceleroTimerEventHandler   // Timer callback handler
    );
    if (AcceleroTimer == NULL)
      {
        SILABS_LOG("Accelero timer create failed");
        appError(APP_ERROR_CREATE_TIMER_FAILED);
      }

    xTimerStart(AcceleroTimer, pdMS_TO_TICKS(1000));
}
```

Figure 84 Firmware controller - InitSensor

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

11.4. AppTask

All the functions are prepared, now we just need to implement them in the AppTask. Before we do that, we need to understand how this task works.

```
108 void AppTask::AppTaskMain(void * pvParameter)
109 {
110     AppEvent event;
111     QueueHandle_t sAppEventQueue = *(static_cast<QueueHandle_t *>(pvParameter));
112
113     CHIP_ERROR err = sAppTask.Init();
114     if (err != CHIP_NO_ERROR)
115     {
116         SILABS_LOG("AppTask.Init() failed");
117         appError(err);
118     }
119
120 #if !(defined(CHIP_DEVICE_CONFIG_ENABLE_SED) && CHIP_DEVICE_CONFIG_ENABLE_SED)
121     sAppTask.StartStatusLEDTimer();
122 #endif
123
124     SILABS_LOG("App Task started");
125
126     while (true)
127     {
128         BaseType_t eventReceived = xQueueReceive(sAppEventQueue, &event, portMAX_DELAY);
129         while (eventReceived == pdTRUE)
130         {
131             sAppTask.DispatchEvent(&event);
132             eventReceived = xQueueReceive(sAppEventQueue, &event, 0);
133         }
134     }
135 }
```

Figure 85 Firmware controller - AppTaskMain

This task continuously listens for events by calling xQueueReceive on the sAppEventQueue. Inside the while (true) loop. It waits indefinitely (portMAX_DELAY) for an event to be received from the event queue. Once an event is received (eventReceived == pdTRUE), it dispatches the event by calling sAppTask.DispatchEvent(&event). It then continues to check for more pending events in the queue by setting eventReceived to the result of xQueueReceive(sAppEventQueue, &event, 0). This is done in a loop to ensure that all pending events are processed without blocking the task. This loop ensures that the task continuously processes events as they arrive, allowing the application to respond effectively to various triggers.

```
126     while (true)
127     {
128         BaseType_t eventReceived = xQueueReceive(sAppEventQueue, &event, portMAX_DELAY);
129         while (eventReceived == pdTRUE)
130         {
131             sAppTask.DispatchEvent(&event);
132             eventReceived = xQueueReceive(sAppEventQueue, &event, 0);
133         }
134     }
135 }
```

Figure 86 Firmware controller - AppTask event loop

Before entering the while (true) loop, the "Init" function is called first.

```
113     CHIP_ERROR err = sAppTask.Init();
114     if (err != CHIP_NO_ERROR)
```


Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

In this function several initializations are done. The most important ones for us are setting the button callback with SetButtonsCb(AppTask::ButtonEventHandler) and initializing the sensor with AcceleroMeter.InitSensor().

```
65  CHIP_ERROR AppTask::Init()
66  {
67      CHIP_ERROR err = CHIP_NO_ERROR;
68
69      chip::DeviceLayer::Silabs::GetPlatform().SetButtonsCb(AppTask::ButtonEventHandler);
70  #ifdef DISPLAY_ENABLED
71      GetLCD().Init((uint8_t *) SENSOR_NAME);
72  #endif
73
74      err = BaseApplication::Init();
75      if (err != CHIP_NO_ERROR)
76      {
77          SILABS_LOG("BaseApplication::Init() failed");
78          appError(err);
79      }
80
81      sSensorLED.Init(SENSOR_LED);
82      sSensorLED.Set(false);
83
84      AcceleroMeter.InitSensor();
```

Figure 87 Firmware controller - AppTask init

The ButtonEventHandler function handles button events. It creates an event object for the button press, specifying which button was pressed and what action was taken. Depending on the button pressed, it assigns a specific handler function to handle the event. Finally, it sends the event to the main task of the application for processing. This ensures that when the user button is pressed, the ActionTriggered function is called to activate the sensor.

```
137 void AppTask::ButtonEventHandler(uint8_t button, uint8_t btnAction)
138 {
139
140     AppEvent button_event           = {};
141     button_event.Type               = AppEvent::kEventType_Button;
142     button_event.ButtonEvent.Action = btnAction;
143
144     if (button == APP_SENSOR_SWITCH)
145     {
146         button_event.Handler = SilabsSensors::ActionTriggered;
147         sAppTask.PostEvent(&button_event);
148     }
149     else if (button == APP_FUNCTION_BUTTON)
150     {
151         button_event.Handler = BaseApplication::ButtonHandler;
152         sAppTask.PostEvent(&button_event);
153     }
154 }
```

Figure 88 Firmware controller - AppTask ButtonEventHandler

After setting up this callback, the sensor is initialized by calling AcceleroMeter.InitSensor(). This function was discussed in section 11.3.. So, after calling this function, the timer for the sensor will be started. Whenever we activate the sensor, this application will make sure that every 30 milliseconds, the data from the accelerometer is placed into the Current-Level attribute.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

11.5. No inertial sensor data in Matter application

When testing the application with the Python test script, I kept receiving a value of 0 for the Current-Level attribute.

Figure 89 Firmware controller - No inertial sensor data

After that I started to debug the application and I noticed that the sensor failed the initialization. I couldn't find any answer as to what the problem might be. Eventually, I connected the logic analyzer to the SPI pins and discovered something strange. When I run the analyzer and start the application, I see that the SPI lines are initialized first (2.5s). Then when the sensor initializes, MOSI (6s) does nothing.

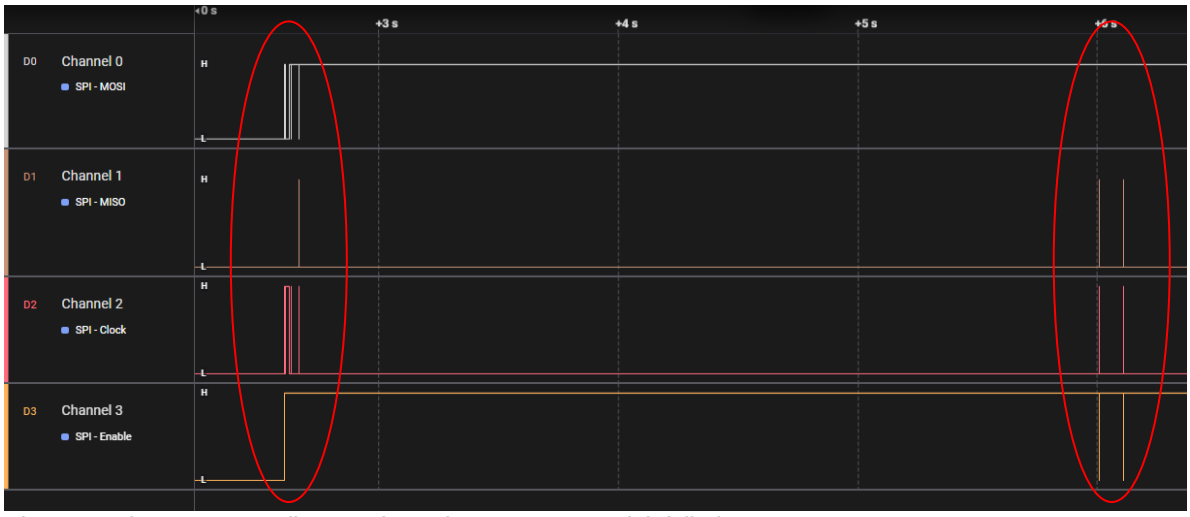


Figure 90 Firmware controller - Logic analyzer SPI sensor init failed

Before we can determine exactly what's wrong, we need to know how the SPI lines should be configured. To do this, I used the bare-metal test, which reads data from the sensor I know works. This way we can measure the SPI lines.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Figure 91 Firmware controller - Logic analyzer SPI sensor working (bare-metal)

Now we know that the MOSI should not be idle high but idle low. This means that the SPI lines are already configured before the sensor is initialized. After searching, there was no configuration in the matter application.

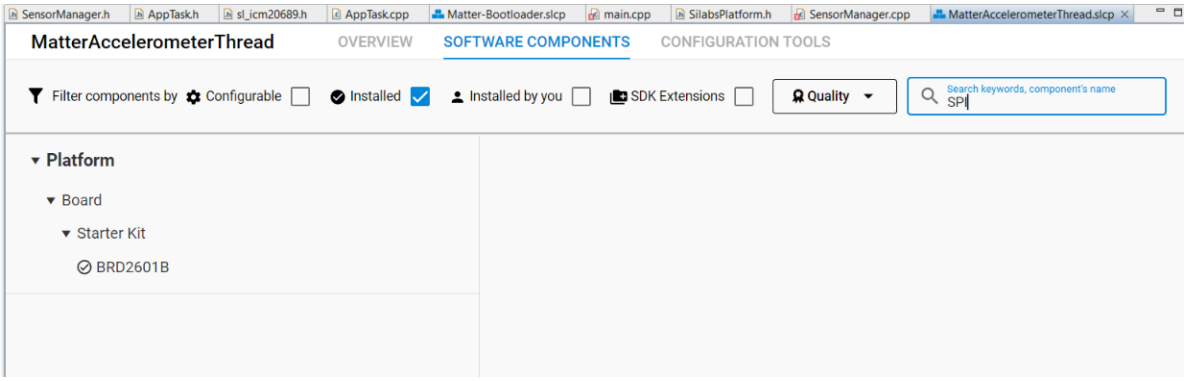


Figure 92 Firmware controller - No SPI configuration

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

11.5.1. Bootloader SPI Flash

Since there was no SPI configuration in the Matter application, I checked the bootloader configuration. I discovered that two SPI-related components are installed in the bootloader. software components.

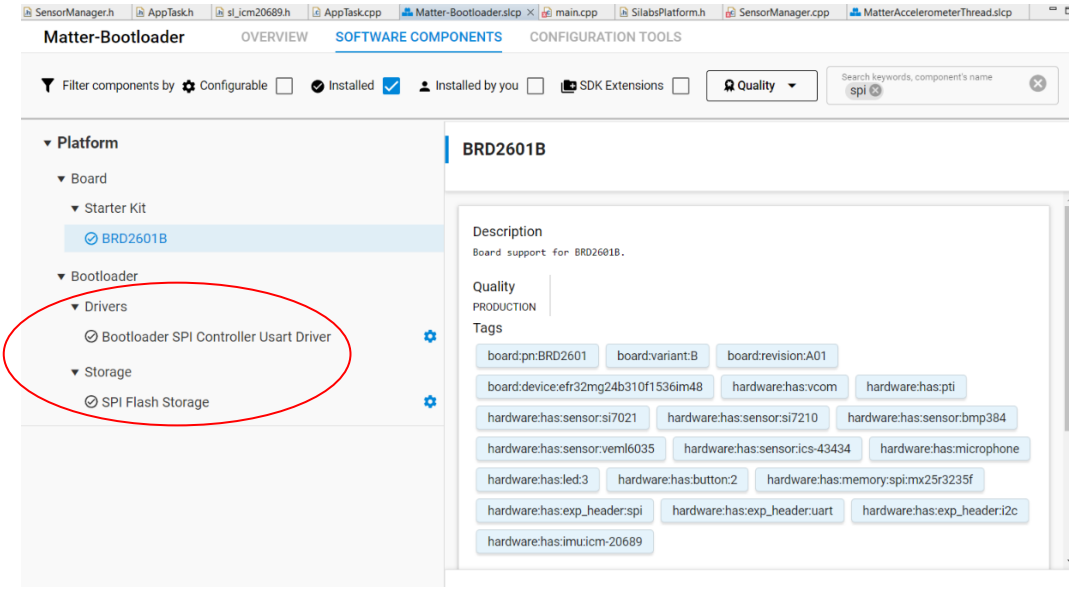


Figure 93 Firmware controller - Bootloader SPI software components

When I read the description of the Bootloader SPI Controller USART Driver, it says The Bootloader SPI Controller USART Driver component provides a simple blocking SPI Universal Synchronous/Asynchronous Receiver/Transmitter (USART) controller implementation for communicating with external devices.

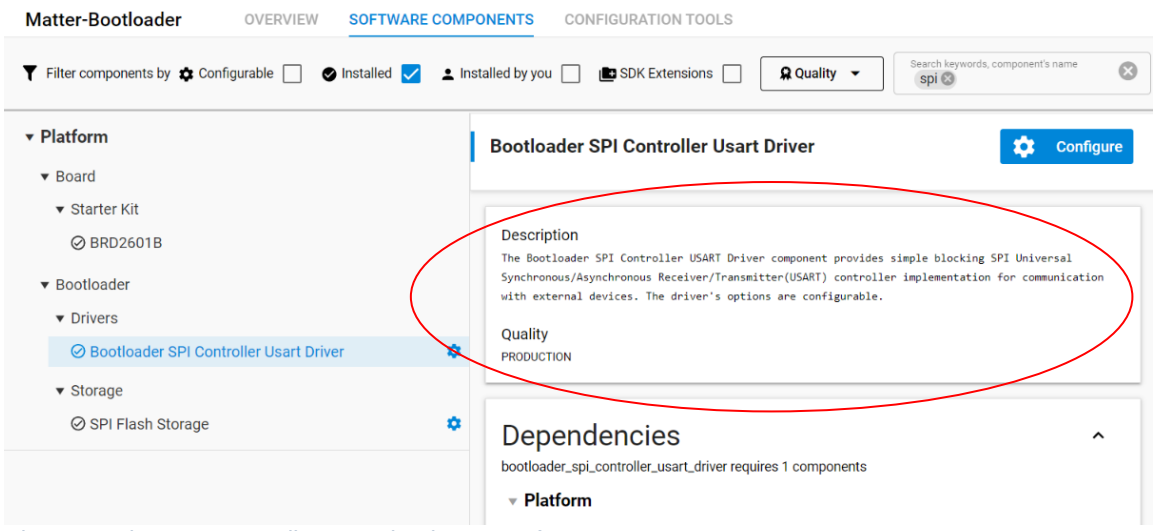


Figure 94 Firmware controller - Bootloader SPI software components

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

So, this component will block the SPI controller. I uninstalled this component and then retested.

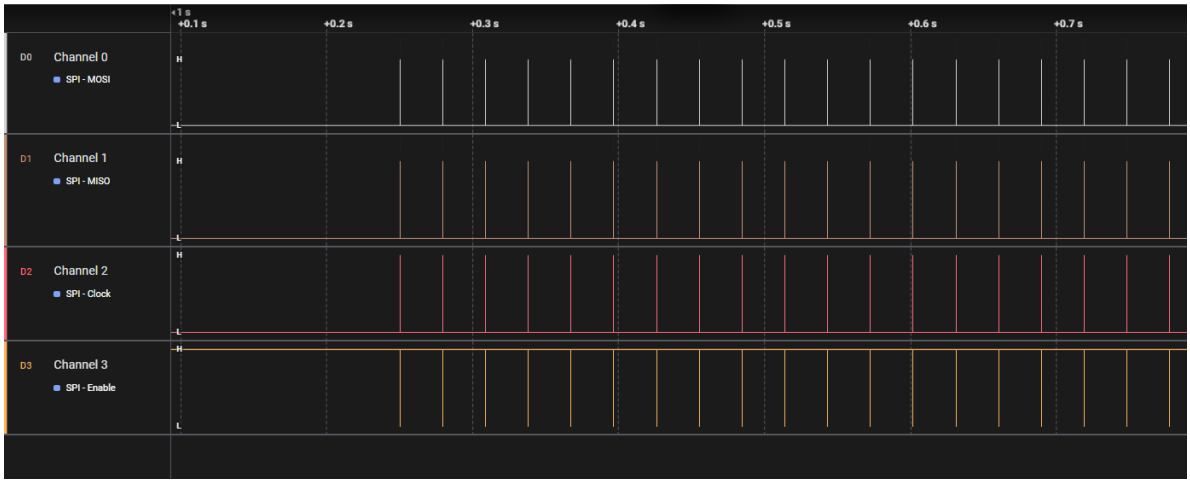


Figure 95 Firmware controller - Logic analyzer SPI sensor working

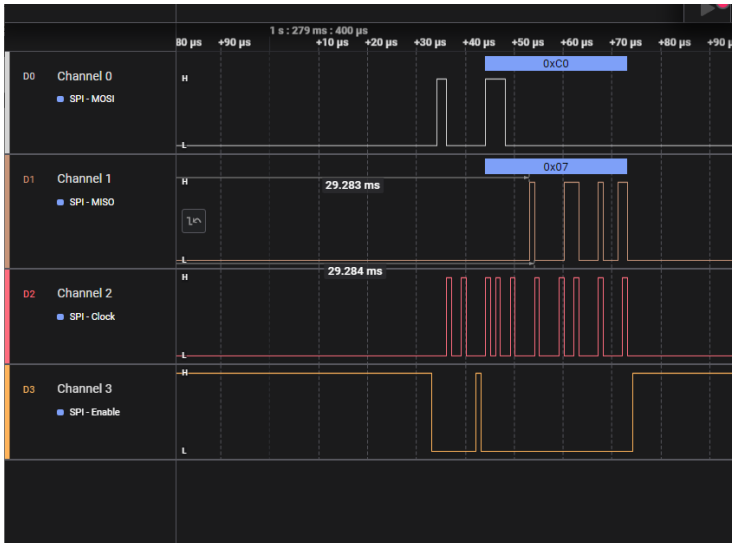


Figure 96 Firmware controller - Logic analyzer SPI sensor working

So now we know that this component was causing the SPI lines to be blocked. When we go back to see if data is coming in, we can see that everything is working as we expected.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

```
[1714668473.416915][3349:3352] CHIP:DL: Long dispatch time: 223 ms, for event type 32784

We reached the end of the startup
['_CHUNK_SIZE', '__class__', '__del__', '__delattr__', '__dict__', '__dir__', '__doc__', '_
'__exit__', '__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '
__iter__', '__le__', '__lt__', '__ne__', '__new__', '__next__', '__reduce__', '__reduce_ex_
tattr__', '__sizeof__', '__str__', '__subclasshook__', '_checkClosed', '_checkReadable', '_
ckWritable', '_finalizing', 'buffer', 'close', 'closed', 'detach', 'encoding', 'errors', 'f
tty', 'line_buffering', 'name', 'newlines', 'read', 'readable', 'readline', 'readlines', 'r
'seekable', 'tell', 'truncate', 'writable', 'write', 'write_through', 'writelines']
63
102
220
231
152
52
42
```

Figure 97 Firmware controller - Inertial sensor data working

12. Breakout game

The Matter Hub has already been prepared as explained in Section 7. Now we need to find a Breakout game. I searched for a Python example for the breakout game on GitHub. I found it at https://github.com/julianolf/pybreakout.git.

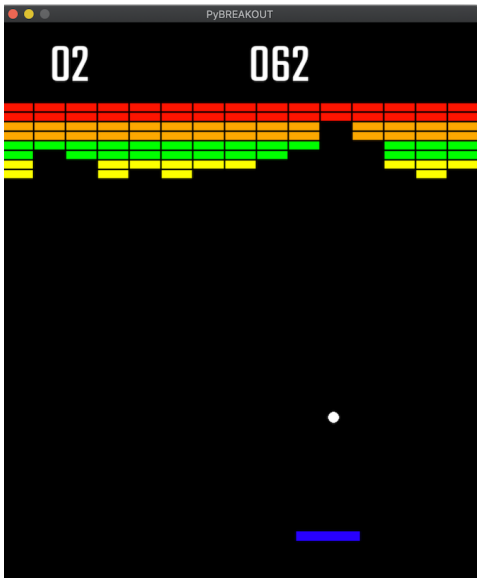


Figure 98 Breakout game - Example

The original game uses the keyboard as input. The goal is to modify this application so that the paddle can be controlled by the sensor. To do this, we need to understand how the application works.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

The game includes three Python files:

- Game.py
- Settings.py
- Sprites.py

In the main application, the starting point is the function main.

```
180  def main():
181      Game().run()
182
```

Figure 99 Breakout game - Main

Then the function “run” from class “Game” gets called.

```
165      def run(self):
166          self.running = True
167          self.start()
168          self.loop()
169          pygame.quit()
170
```

Figure 100 Breakout game - Run

Within this there is a function called "loop". This function will continuously update changes and then draw it to the frame. I don't need to know more, I just need to know where variables change.

```
114      def loop(self):
115          while self.running:
116              self.clock.tick(settings.FPS)
117              self.update()
118              self.draw()
119              self.events()
120
```

Figure 101 Breakout game - Loop

So with "update" we can find the function that moves the paddle. The function "update" refers to another object function "sprites.update".

```
95
96      def update(self):
97          self.sprites.update()
```

Figure 102 Breakout game - Update

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

Important for now, there are several functions called "update" in the object sprites. All these functions are called because multiple sprite objects are grouped.

```
11  class Game:
12      def __init__(self):
13          pygame.init()
14          pygame.mixer.init()
15          pygame.display.set_caption(settings.TITLE)
16          self.screen = pygame.display.set_mode(settings.WIN_SIZE)
17          self.clock = pygame.time.Clock()
18          self.sprites = pygame.sprite.Group()
19          self.bricks = pygame.sprite.Group()
20          self.sfx = {
21              sound: pygame.mixer.Sound(path.join(settings.SFX, f'{sound}.wav'))
22              for sound in ('bounce', 'explosion', 'launch', 'level')
23          }
```

Figure 103 Breakout game - Sprite objects grouped

If we look in sprites.py, we find a class called "Paddle". In this class everything related to the paddle is bundled, including how it is moved. This function asks if the left or right key is pressed, then it calls the function "left" or "right

```
59      def update(self):
60          keys = pygame.key.get_pressed()
61          if keys[pygame.K_LEFT]:
62              self.left()
63          if keys[pygame.K_RIGHT]:
64              self.right()
```

Figure 104 Breakout game - Paddle update

The function “left” and "right" will set the position of the paddle with rect.x.

```
40      def left(self):
41          self.rect.x -= self.velocity
42          if self.rect.left <= 0:
43              self.rect.left = 0
44
45      def right(self):
46          self.rect.x += self.velocity
47          if self.rect.right >= settings.WIDTH:
48              self.rect.right = settings.WIDTH
49
```

Figure 105 Breakout game - Paddle left & right

12.1. Changes

Now that we know how the paddle is controlled, we can make some changes. We just need to make sure that the paddle is controlled by an input we get from the level control script explained in section 8.3. So we need to create a function that controls the paddle, and we need to integrate the script we created into the main application. However, we have to be careful because we don't want it to slow down the game. Therefore, we need to make sure that the script runs within the application, but does not affect and slow down the application. The solution to this is to use threading.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

12.1.1. Move_to_pos

To control the paddle, I created a function called "move_to_pos". This function will set the position of the paddle with the given argument "pos". So we can call this function to move the paddle to a position we want.

```
52      #This function makes it possible that the controller can set the position of the paddle
53      def move_to_pos(self, pos):
54          self.rect.x = pos
55          if self.rect.left <= 0:
56              self.rect.left = 0
57          if self.rect.right >= settings.WIDTH:
58              self.rect.right = settings.WIDTH
```

Figure 106 Breakout game - move_to_pos

12.1.2. Threading

To ensure that the script that reads data from the game controller does not interfere with the game itself, we will use threading. A thread is a separate execution flow. This means that the application will have two things happening at the same time, the game and the script.

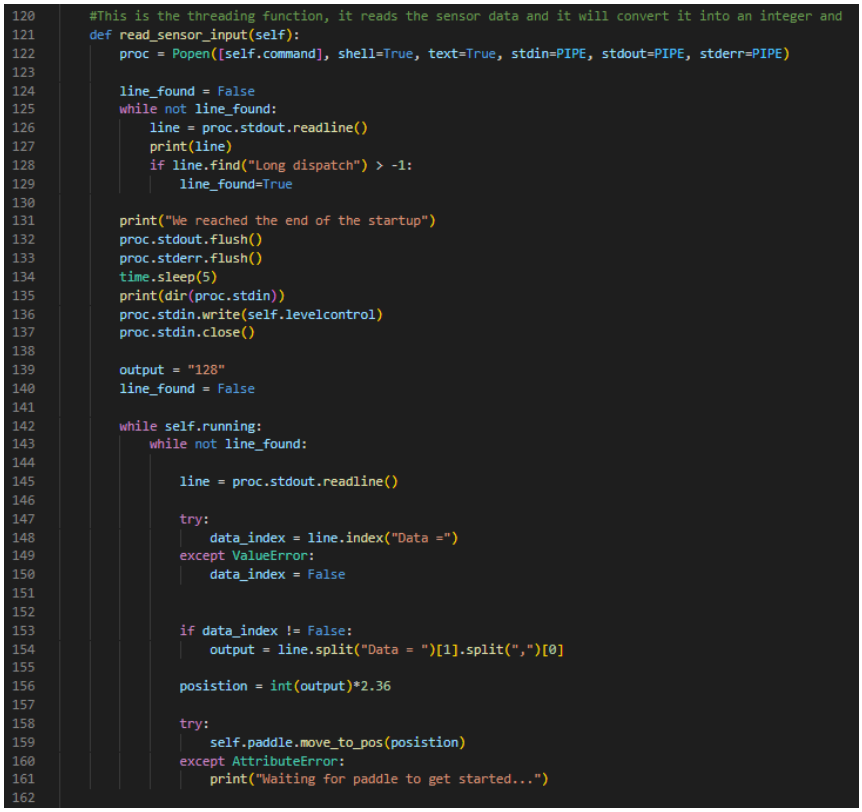
NodeId, as in the test, will retain the ID that was assigned after the sensor was commissioned. This ID must be passed as an argument when starting the application. Then we have the command to start the interactive mode and the command to subscribe to the Current-Level attribute of the level control cluster.

```
11  class Game:
12      def __init__(self):
13          self.nodeId = sys.argv[1]
14          self.command = "/home/ubuntu/scripts/matterTool.sh interactive start"
15          self.levelcontrol = "levelcontrol subscribe current-level 0 1 " + str(self.nodeId) + " 2\n"
```

Figure 107 Breakout game - CHIP-tool commands

In order to create the thread, we need a function that will run as a thread. In this function, named "read_sensor_input", we will place the script used in section 8.3..

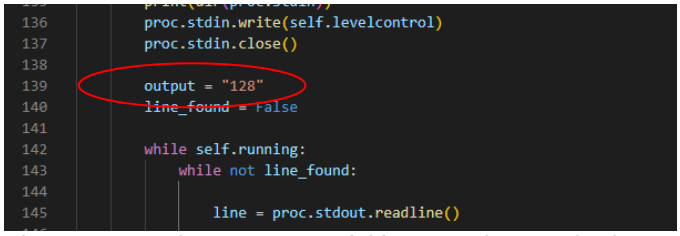
Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08



```
120     #This is the threading function, it reads the sensor data and it will convert it into an integer and
121     def read_sensor_input(self):
122         proc = Popen([self.command], shell=True, text=True, stdin=PIPE, stdout=PIPE, stderr=PIPE)
123 
124         line_found = False
125         while not line_found:
126             line = proc.stdout.readline()
127             print(line)
128             if line.find("Long dispatch") > -1:
129                 line_found=True
130 
131         print("We reached the end of the startup")
132         proc.stdout.flush()
133         proc.stderr.flush()
134         time.sleep(5)
135         print(dir(proc.stdin))
136         proc.stdin.write(self.levelcontrol)
137         proc.stdin.close()
138 
139         output = "128"
140         line_found = False
141 
142         while self.running:
143             while not line_found:
144 
145                 line = proc.stdout.readline()
146 
147                 try:
148                     data_index = line.index("Data =")
149                 except ValueError:
150                     data_index = False
151 
152 
153                 if data_index != False:
154                     output = line.split("Data = ")[1].split(",")[0]
155 
156                 posistion = int(output)*2.36
157 
158                 try:
159                     self.paddle.move_to_pos(posistion)
160                 except AttributeError:
161                     print("Waiting for paddle to get started...")
162 
```

Figure 108 Breakout game - Thread function read_sensor_input

A few changes have been made to the script. A variable "output" has been created, this variable will contain the filtered data.



```
136         proc.stdin.write(self.levelcontrol)
137         proc.stdin.close()
138 
139         output = "128"
140         line_found = False
141 
142         while self.running:
143             while not line_found:
144 
145                 line = proc.stdout.readline()
```

Figure 109 Breakout game - Variable output (sensor data)

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

after the filtered data is saved to "output", it is converted to an integer and multiplied by '2.36'. This adjustment is made because the data originally comes from the sensor as an 8-bit value, ranging from 0 to 255, while the maximum range of the paddle is 600.

```
152
153                 if data_index != False:
154                     output = line.split("Data = ")[1].split(",")[0]
155
156                 posistion = int(output)*2.36
157
158                 try:
159                     self.paddle.move_to_pos(posistion)
160                 except AttributeError:
161                     print("Waiting for paddle to get started...")
162
163
```

Figure 110 Breakout game - sensor data adjustment

Then we will pass the converted value as an argument to the function "move_to_pos" that we created earlier.

```
152
153                 if data_index != False:
154                     output = line.split("Data = ")[1].split(",")[0]
155
156                 posistion = int(output)*2.36
157
158                 try:
159                     self.paddle.move_to_pos(posistion)
160                 except AttributeError:
161                     print("Waiting for paddle to get started...")
162
163
```

Figure 111 Breakout game - move_to_pos with argument

Now that the function is created, the last step is to create and start the thread. This is done in the run function and before the loop is started.

```
163
164     def run(self):
165         self.running = True
166         self.start()
167
168         #Threading was needed to make sure reading the sensor is not interrupting the game speed
169         x = threading.Thread(target=self.read_sensor_input)
170         x.start()
171
172         #Sleep was needed to make sure the thread is starting and make no exceptions - time can be modifie
173         time.sleep(20)
174
175         self.loop()
176         pygame.quit()
177
```

Figure 112 Breakout game - Create and start Thread

This way, the script has absolutely no effect on the speed of the game. The game runs smoothly, and the paddle moves exactly as expected.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

13. Conclusion

This has been an incredibly enjoyable project to work on, and I've learned a lot from it. I've gained extensive knowledge about the Matter protocol and all aspects surrounding it, including its purpose, structure, capabilities, and limitations. One of the technical skills I've certainly acquired here is problem solving. Throughout this project, there have been many obstacles. It started with the realization that what we wanted to achieve was "officially" not possible, but we decided to go ahead anyway. It certainly wasn't easy, but I learned a lot from it. I was able to overcome all the obstacles and successfully complete the project in the end.

Additionally, this internship was successful for me in several ways. It sparked my interest in embedded software and demonstrated that every project has numerous obstacles that need to be solved. I also learned a lot about Python, especially how to access the shell through Python to run commands and extract data from it. Working with existing software required me to analyze a lot of code, which also taught me a great deal.

Moreover, I gained valuable experience in project planning through the use of Gantt charts, learned how to make technical analyses, and got a firsthand look at what it's like to work within an R&D department.

I would like to extend my gratitude to Dekimo for providing this opportunity. Their support and guidance made this learning experience invaluable.

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08

14. Document Information

14.1. Version History

Date	Author	Version	Description
22/09/2023	BGY	0.1	Project overview and wireless controller description
21/03/02024	BGY	0.2	Sections Thread and Matter
05/04/2024	BGY	0.3	Gecko SDK, inertial sensor, tests
06/06/2024	BGY	1.0	Official release

14.1. Related References and Documents

Ref	Description
20231201 - Planning Matter Project.docx	Project planning document.
Matter-1.2-Core-Specification.pdf	Full specification document of the Matter protocol.
Matter-1.2-Application-Cluster-Specification.pdf	Supported application cluster specification.
Matter-1.2-Device-Library-Specification.pdf	All the device types are defined in this document.
DS-000143-ICM-20689-TYP-v1.1.pdf	The ICM-20689 motion tracking sensor datasheet.
https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/protocols/matter/index.html	Matter explanation from Nordic Semiconductor.
https://docs.silabs.com/matter/latest/matter-start/	Matter explanation from Silicon Labs.
https://csa-iot.org/	Connectivity Standard Alliance

Dekimo Turnhout
Slachthuisstraat 112/4
B-2300 Turnhout
Tel: +32(0)14 63 89 08