**Technical & Functional analysis**

**Breakout game with gyroscope controller using Thread/Matter**

|  |  |
| --- | --- |
|  |  |

Date : 3/7/2024

Contacts : Dekimo : Gysen Bart

Table of Contents

[1. Project overview 4](#_Toc160713565)

[1.1. Matter protocol 4](#_Toc160713566)

[1.2. OpenThread network protocol 4](#_Toc160713567)

[2. Game controller 5](#_Toc160713568)

[2.1. EFR32MG24 - BRD2601B Dev Kit 5](#_Toc160713569)

[2.2. EFR32MG24 Wireless SoC 6](#_Toc160713570)

[2.3. ICM-20689 6-Axis sensor 7](#_Toc160713571)

[2.4. User Button 8](#_Toc160713572)

[2.5. HAL Config 9](#_Toc160713573)

[3. (Open) Thread 9](#_Toc160713574)

[3.1. OpenThread Border Router 10](#_Toc160713575)

[3.2. Rights – OpenThread – Thread 11](#_Toc160713576)

[4. Raspberry Pi 4 11](#_Toc160713577)

[5. Matter protocol 12](#_Toc160713578)

[5.1. Terms 12](#_Toc160713579)

[5.2. About 12](#_Toc160713580)

[5.3. Documents 12](#_Toc160713581)

[5.4. Architecture 13](#_Toc160713582)

[5.4.1. Network 13](#_Toc160713583)

[5.4.2. Device Data Model 17](#_Toc160713584)

[5.4.3. Interaction Model 20](#_Toc160713585)

[5.4.4. Read Interaction 22](#_Toc160713586)

[5.4.5. Subscription Interaction 23](#_Toc160713587)

[5.4.6. Write Interaction 24](#_Toc160713588)

[5.5. Matter Hub & Brigde 24](#_Toc160713589)

[5.6. Certification 24](#_Toc160713590)

[6. Silabs Gecko SDK V4.3 24](#_Toc160713591)

[6.1. Locations 24](#_Toc160713592)

[6.2. Custom clusters 24](#_Toc160713593)

[7. ICM-20689 Motion tracking sensor 24](#_Toc160713594)

[7.1. Using the ICM-20689 driver 25](#_Toc160713595)

[7.2. Tests 27](#_Toc160713596)

[7.2.1. Raw data 27](#_Toc160713597)

[7.2.2. Converted data 29](#_Toc160713598)

[8. Issues 29](#_Toc160713599)

[8.1. Gyroscope sensor problem 29](#_Toc160713600)

[8.1.1. Possible solution 29](#_Toc160713601)

[8.2. Connection between matter device (using chip-tool) and the “game” 30](#_Toc160713602)

[8.2.1. Possible solution 30](#_Toc160713603)

[9. Tests 30](#_Toc160713604)

[9.1. Simple on/off device 30](#_Toc160713605)

[9.2. Level control test 33](#_Toc160713606)

[9.2.1. Capture data using chip-tool 35](#_Toc160713607)

[9.3. Terminal output test 36](#_Toc160713608)

[10. Application 36](#_Toc160713609)

[11. Notes 37](#_Toc160713610)

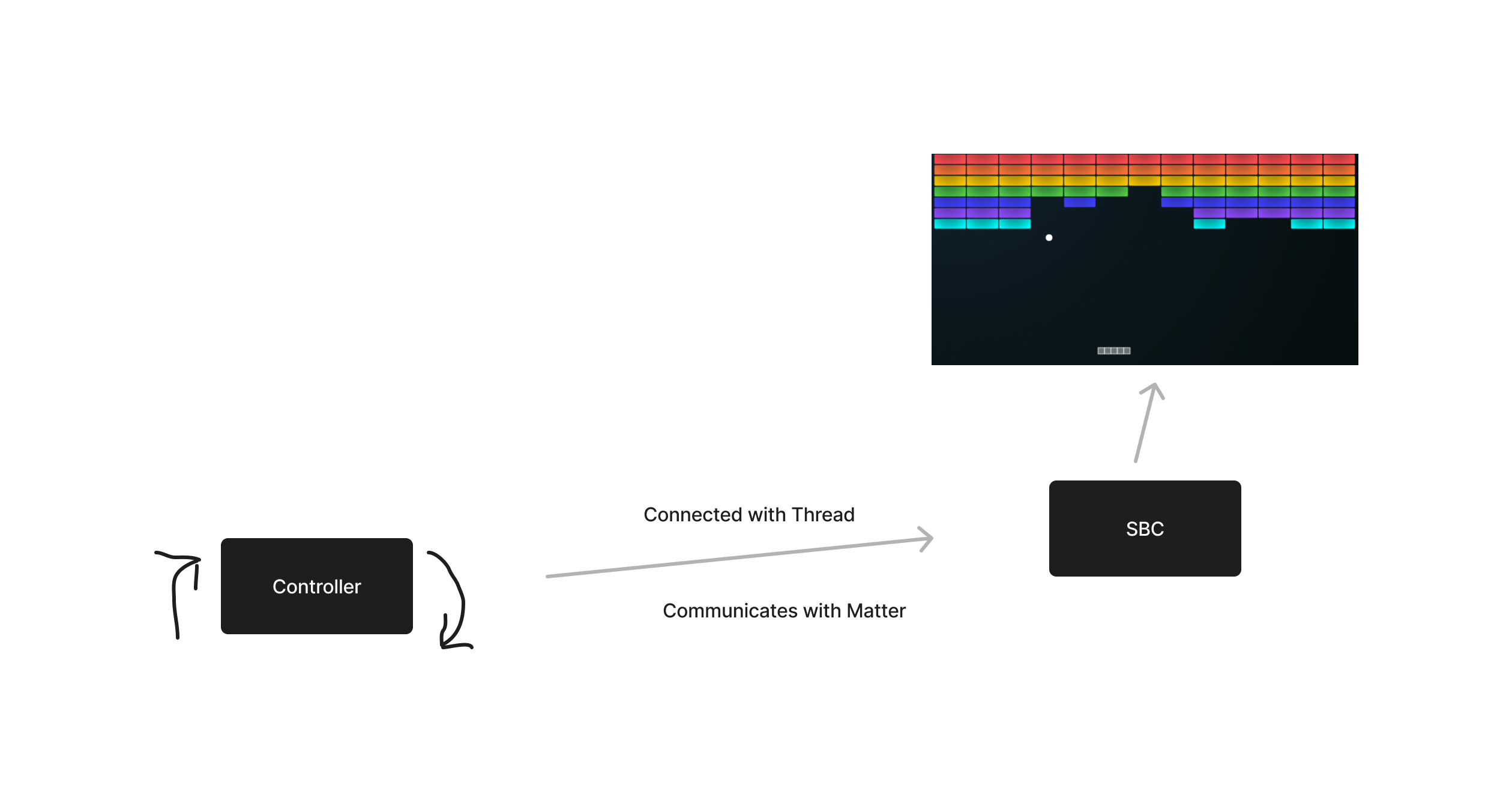
[12. Document Information 38](#_Toc160713611)

[12.1. Version History 38](#_Toc160713612)

[12.1. Related Documents 39](#_Toc160713613)

# 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.



## 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.

## OpenThread network protocol

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

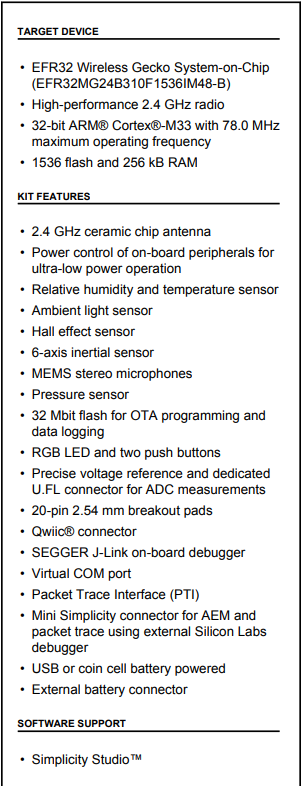
# Game controller

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

## EFR32MG24 - BRD2601B Dev Kit

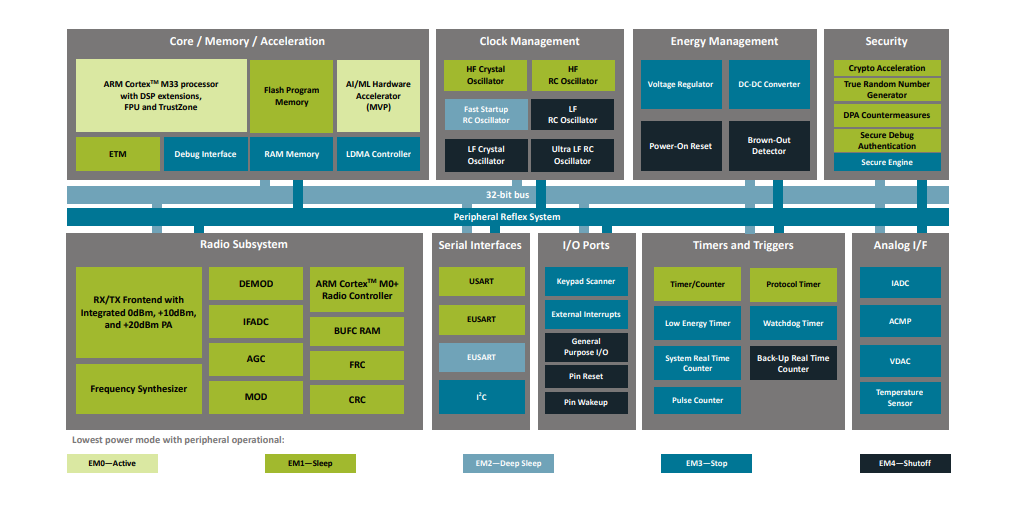
The EFR32MG24 Dev Kit board (BRD2601B) supports al the elements that is necessary for the controller. So there is no need to develop a PCB with al this components.

## EFR32MG24 Wireless SoC

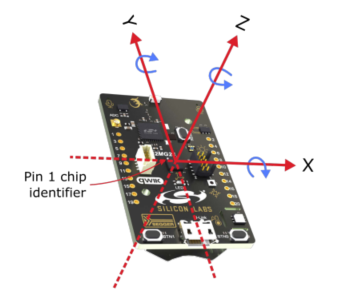
The development board uses the EFR32MG24 wireless SoC (System on a Chip). This SoC is ideal for meshing wireless solutions using Matter and OpenThread. It provides al the built in features that are relevant for this project, like:

* 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.

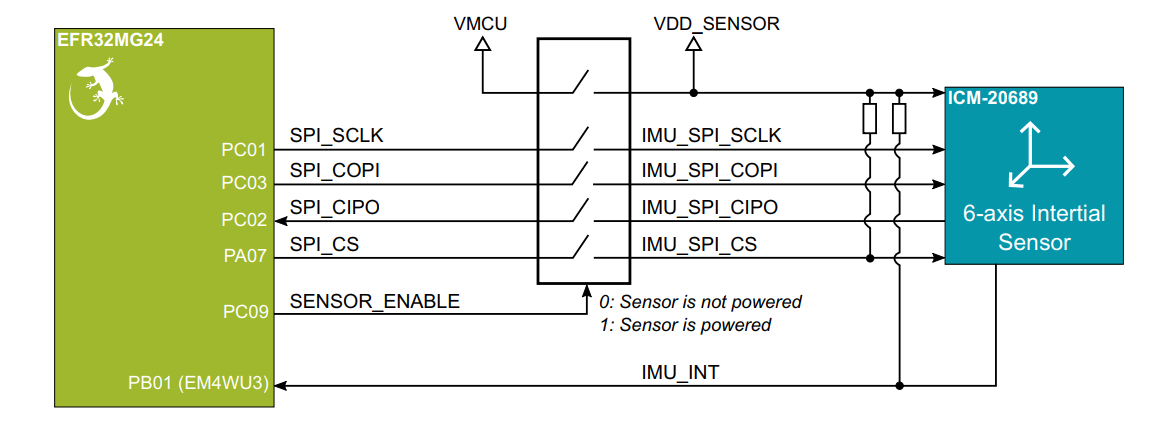


## ICM-20689 6-Axis sensor

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.

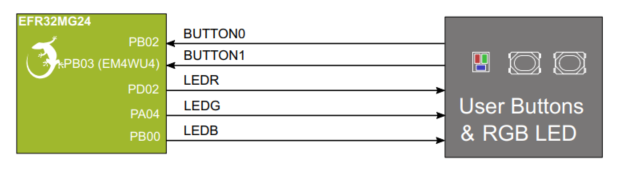


In the development board the sensor is connected an communicates over SPI. The SPI lines are interrupted through a switch to prevent power consumption when not used. Before the sensor can be used in the application it must be enabled by setting PC09 high.

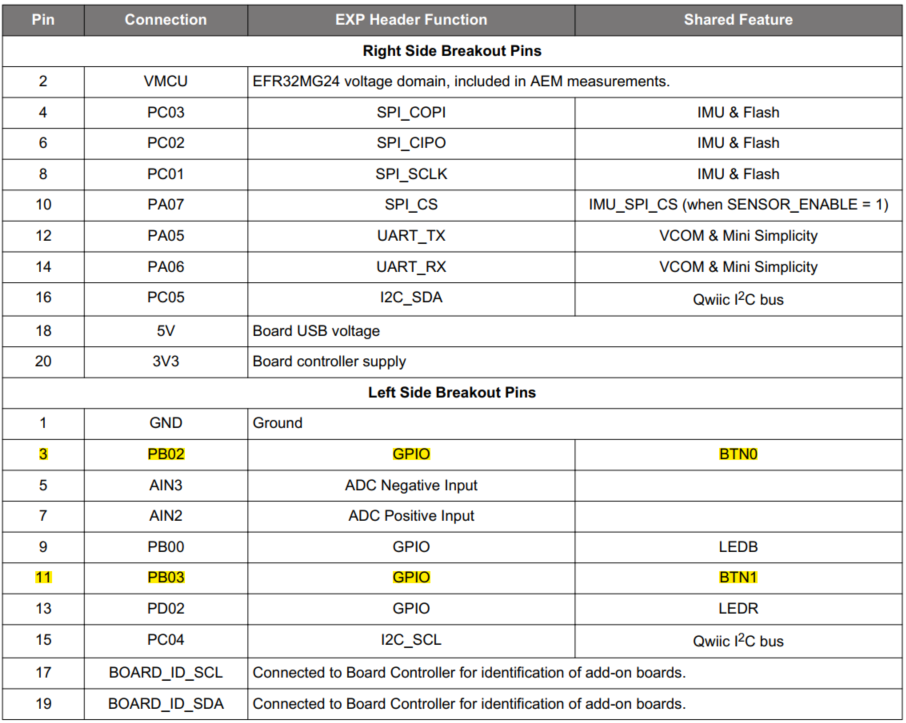


## User Button

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



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.



## HAL Config

In the Matter project you can find the configuration file where all the onboard sensors is listed in. you can find this under config/sl\_board\_control\_config.h.

~~To enable the gyroscope we have to set the define SL\_BOARD\_ENABLE\_SENSOR\_IMU~~

To enable the gyroscope we have to install the software component in the gecko SDK.

# (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.

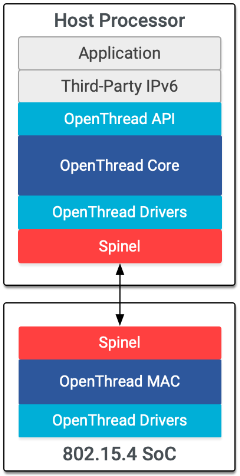
Thread uses the same technology as Zigbee (IEEE 802.15.4) but provides IP connectivity similar to Wi-Fi. Unlike Zigbee, Thread by itself does not allow controlling devices: It is just a communication protocol. To control the Thread devices, a higher-level protocol is required e.g. Matter. Thread devices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

**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.

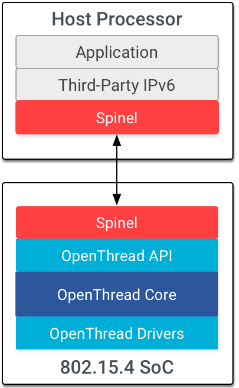
OpenThread released by Google is an open-source implementation of Thread®. Google has released OpenThread to make the networking technology used in Google Nest products more broadly available to developers, in order to accelerate the development of products for the connected home and commercial buildings.

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.

**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.

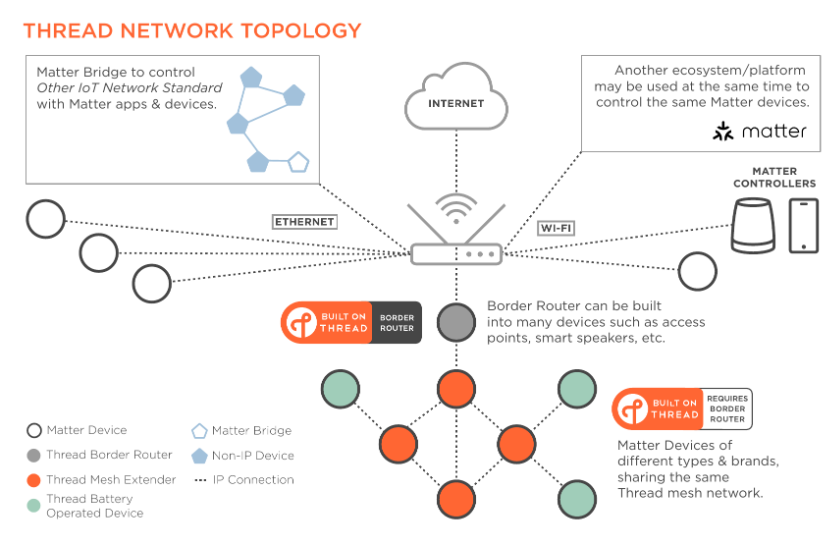
**NCP**



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.

## 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. It does not look at the content of these packets, it just forwards them.



## 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.

# 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.

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

# Matter protocol

## Terms

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

|  |  |
| --- | --- |
| **Term** | **Definition** |
| Device | A piece of equipment containing one or more Nodes |
| Node | An addressable entity which supports the Matter protocol stack and has its own Operational Node ID and Node Operational credentials. A Device may host multiple Nodes. |
| 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. |
| Cluster |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 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.

## Documents

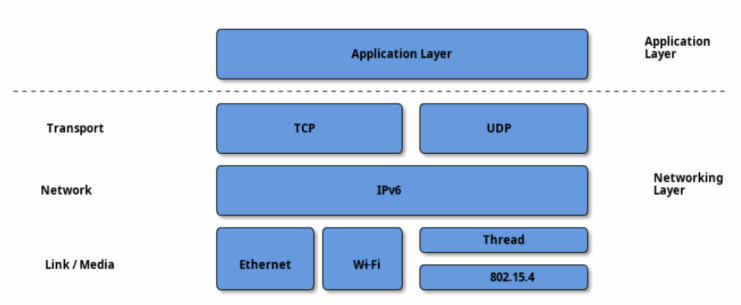
The Matter protocol is specified in Three main documents and can be downloaded from the CSA website. These documents are necessary when developing a Matter product. With every update these documents changes so make sure when developing a product you get the latest documents.

* **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.

## Architecture

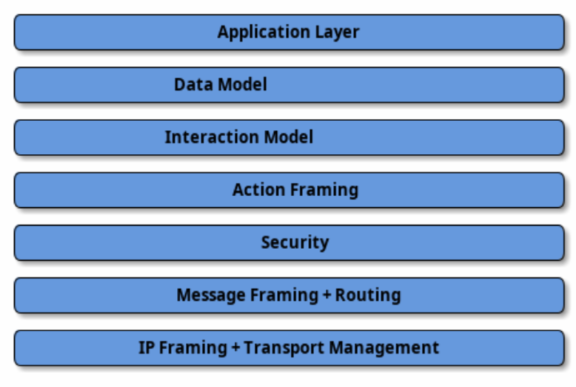
### Network

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.



Matter Protocol 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.



* **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.

**Network Topology**

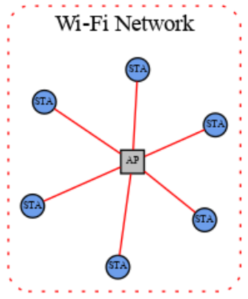
In principle, any IPv6 network is suitable for Matter deployment butt the focus is on three link layer technologies: Ethernet, Wi-Fi and Thread. Matter treats networks as shared resources: it makes no stipulation of exclusive network ownership or access. As a result, it is possible to overlay multiple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

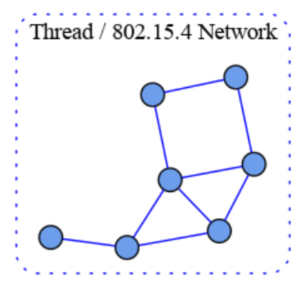
This protocol may operate in the absence of globally routable IPv6 infrastructure. This requirementenables operation in a network disconnected or firewalled from the global Internet. It also enablesdeployment in situations where the Internet Service Provider either does not support IPv6 on consumer premises 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**

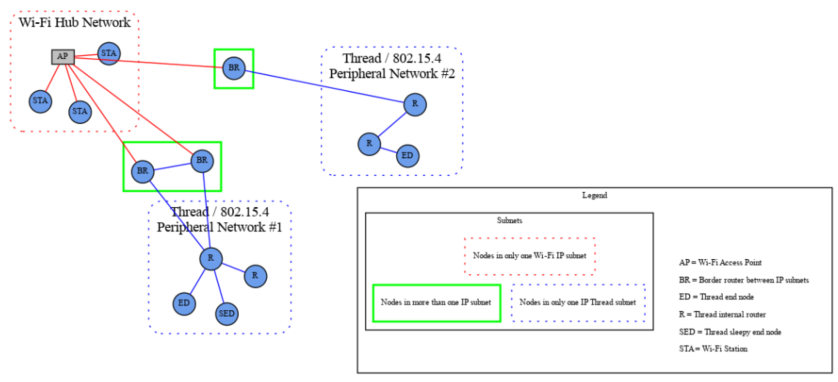
In the single network topology, all Matter devices are connected to a single network. This could by a Thread network, Wi-Fi or Ethernet network. In the case of WiFi/Ethernet, the network could in fact span multiple Wi-Fi and/or Ethernet segments provided that all the segments are bridged at the link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





**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.



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.

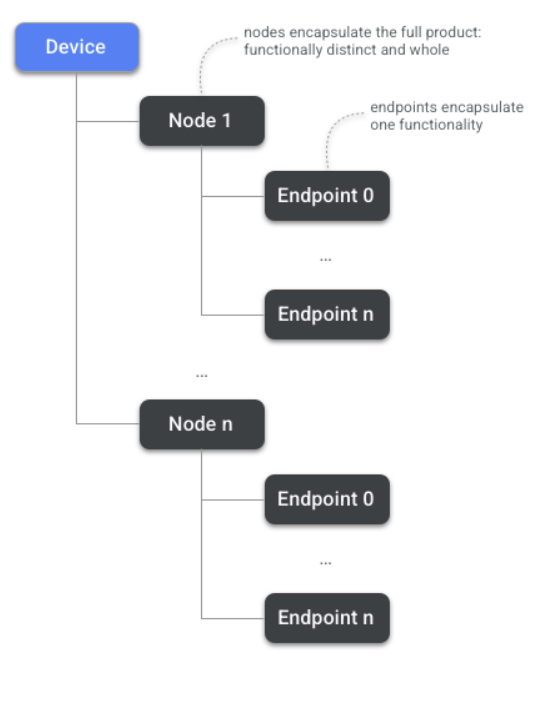
Regardless of the Network topology being used, Matter has a concept of Fabrics. A Matter Fabric is a security domain that contains a collection of nodes. These nodes can be identified and can communicate with each other within the context of that security domain. Each Matter Fabric has a unique Node 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.

### Device Data Model

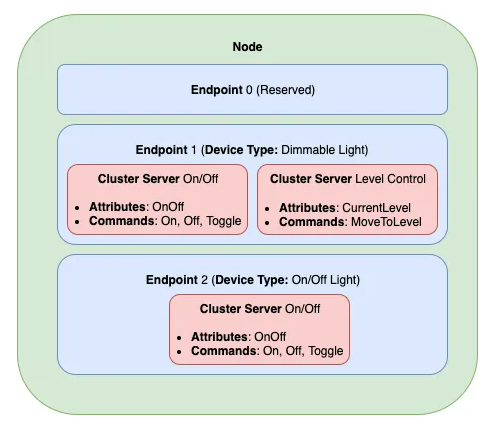
The data model in Matter is a hierarchical modeling of a devices features, including nodes, endpoints, clusters and device types where the node is the highest level data element. A single device can be represented by one or more nodes. An environment where multiple Matter nodes interoperate is referred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



**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.



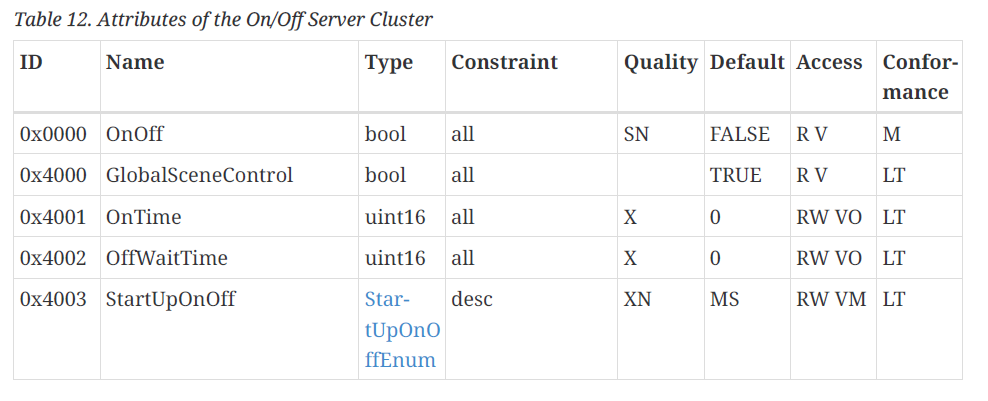
**Endpoint**

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.

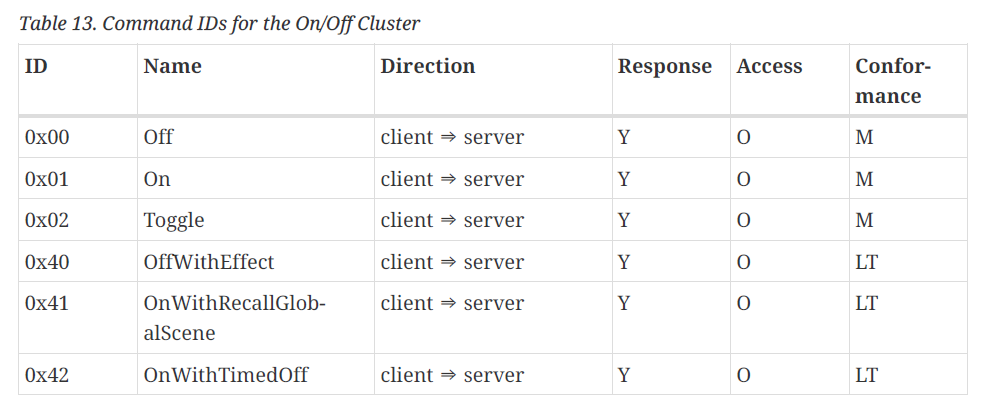
**Clusters**

A cluster groups together commonly used functionality in a reusable building block. Endpoint 1 has two cluster 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.



* **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.



* **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.

It is possible to make a 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.

**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.

**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

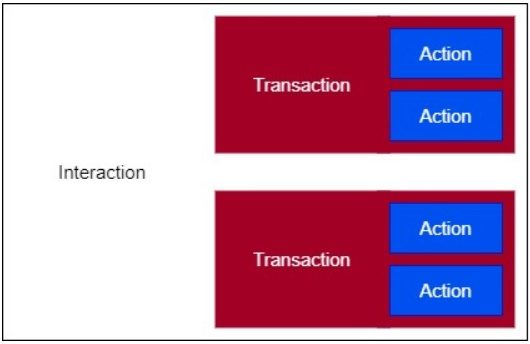
All the device types are defined in the **Matter-(version)-Application-Cluster-Specification.pdf** document.

**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.

### 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.



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.
* **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.

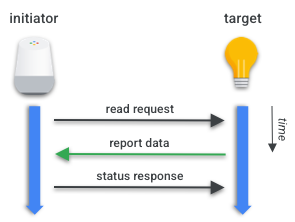
* **Initiators and Targets**: Interactions happen between initiator and target nodes. The initiator starts the transaction and the target responds. Usually the client cluster is the initiator node and de server the target node.
* **Transaction ID**: The transaction ID field must be present in all actions 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:

****

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.

### Read Interaction

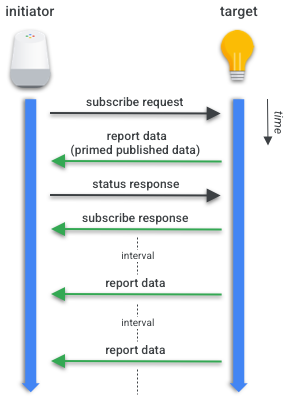
When an initiator wishes to determine the value of one or more attributes or events on a node, an Read Interaction is generated. In this interaction the following actions occur:



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

### 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 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 Actions containing the first batch of reported data, the pimed 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.

### 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.

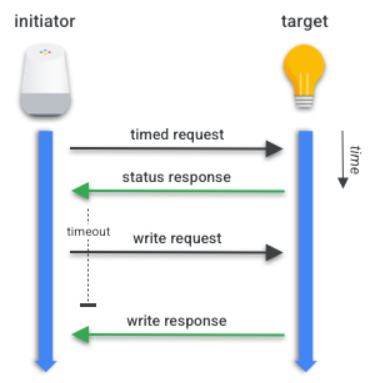
**Untimed Write Transaction**

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 wil respond with the Write Response Action containing:
  + **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.

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

## Matter Hub & Brigde

Vertel wat dit is en waarom we dit nodig hebben.

## Matter production guide

Explained in the silabs-matter-expl.pdf page 314.

# Silabs Gecko SDK V4.3

**This documentation is based on the Simplicity Studio v5 IDE!!**

## Locations

* Endpoint configuration file can be found in the project under Workspace/projectfolder/autogen/zap-generated/endpoint\_config.h

## Custom clusters

There is an possibility to add a custom Cluster. The best way to do this is to modify an existing cluster xml file. Clusters xml files can be find in:

* **If using simplicity studio and using the Gecko SDK:** path\_to\SimplicityStudio\SDKs\gecko\_sdk\extension\matter\_extension\src\app\zap-templates\zcl\data-model\chip

# ICM-20689 Motion tracking sensor

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

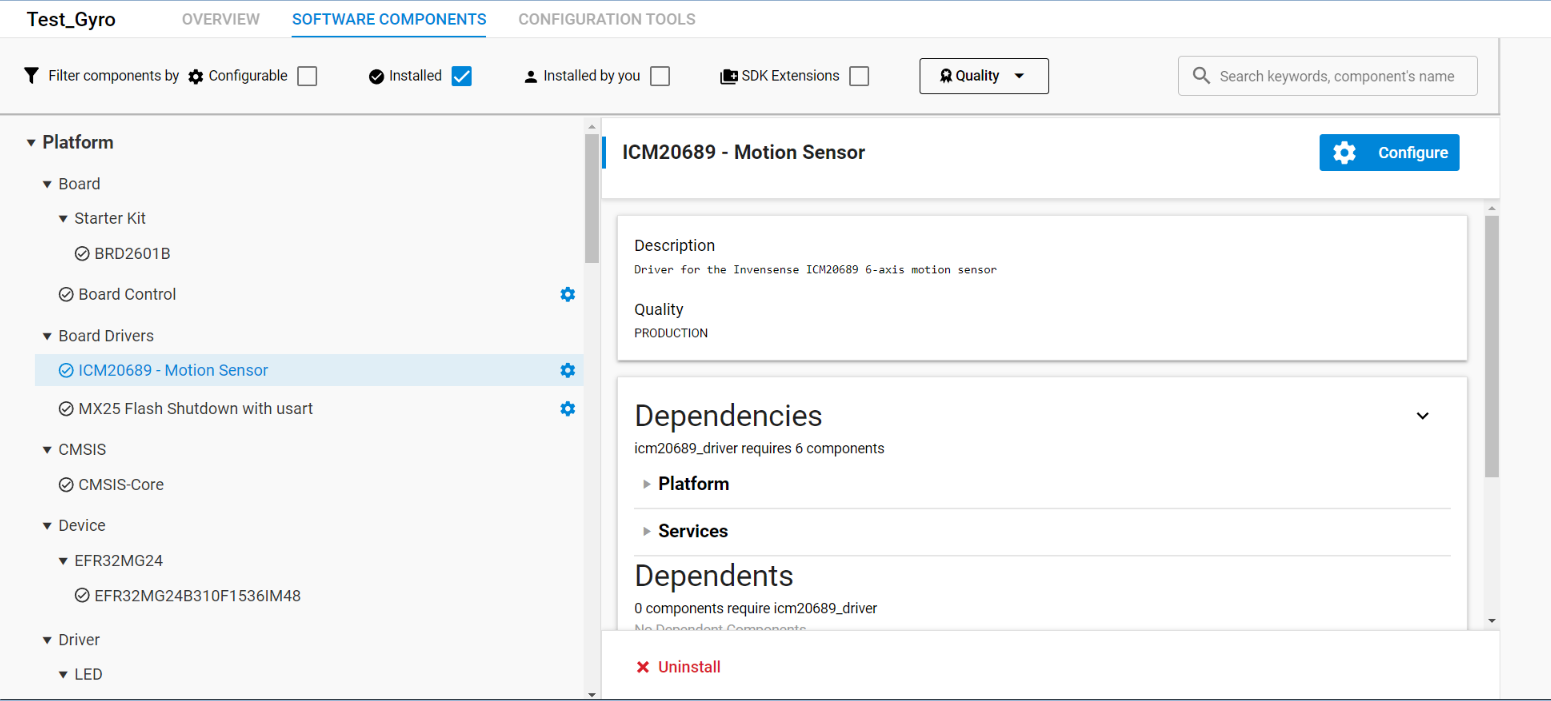
For this project, the gyroscope, as well as the accelerometer, can be used, but there are two differences. The gyroscope measures the speed (°/s), so when moving the sensor give’s a value how fast the sensor moves. When the sensor stops moving we get a “0” value.

The accelerometer measures the gravity (g). The sensor holds 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 certain value.

## Using the ICM-20689 driver

The driver contains all the functions needed to use the sensor.

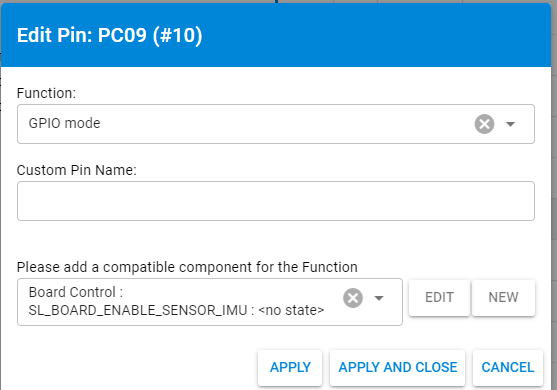
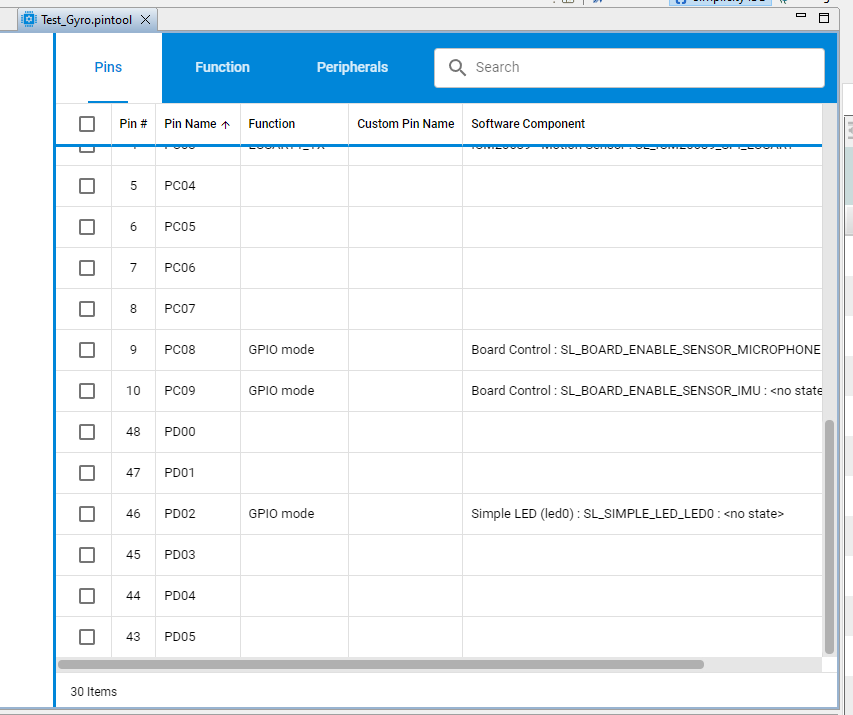
Silicon Labs has a driver for this sensor that contains all the functions needed to use the sensor. The driver can be used by installing the software component in simplicity studio. Open the project file (.slcp) and select software components. Search for ICM20689 and install the component.



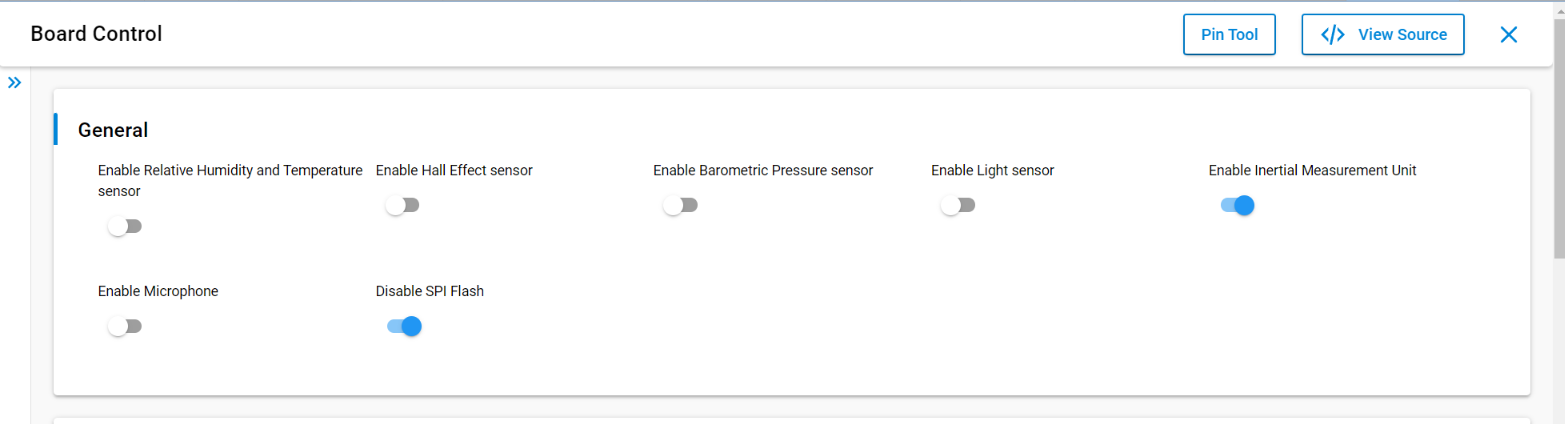
After installed the driver set PC09 to high to enable the sensor, refer to [2.3 ICM-20689 6-Axis sensor](#_ICM-20689_6-Axis_sensor). There are two options to do this, by using the pin tool or manualy.

* By using the pintool:

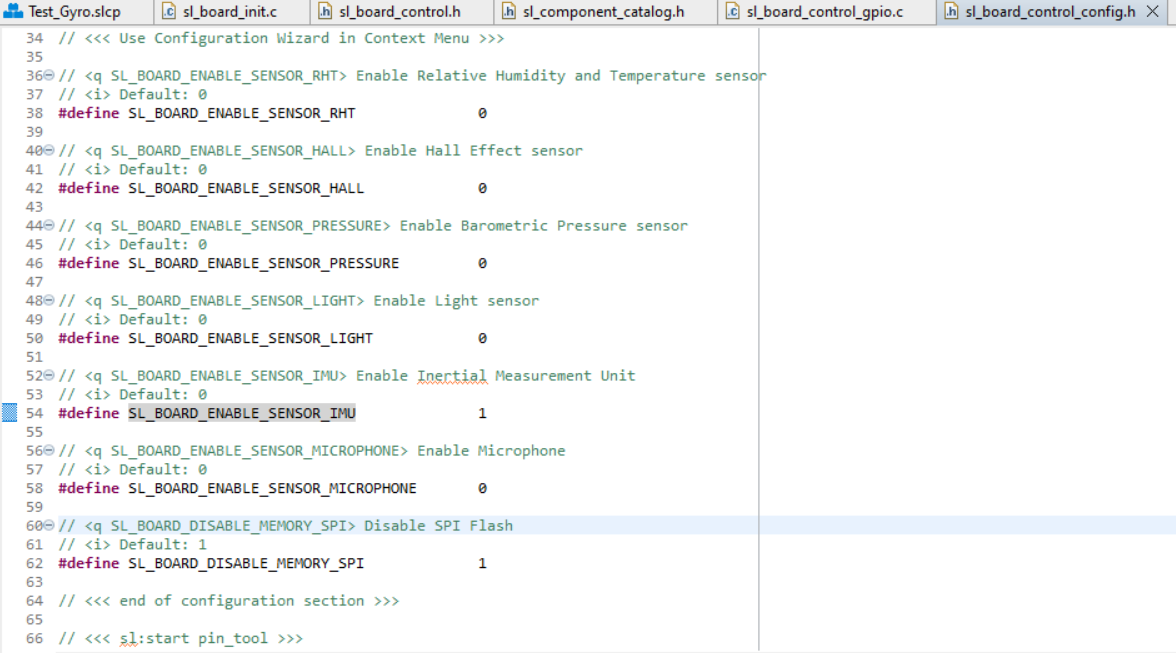
Select PC09, then click on edit



After that click on Enable Inertial Measurement Unit.



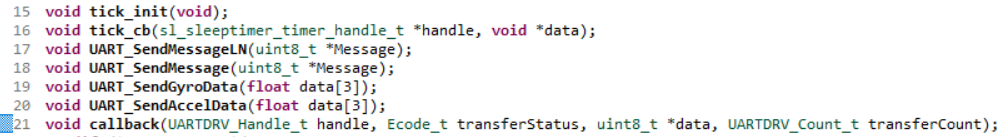
* Manually:  
  To enable the sensor go to the folder config in the project and open sl\_board\_control\_config.h and set the define SL\_BOARD\_SENSOR\_IMU to 1.



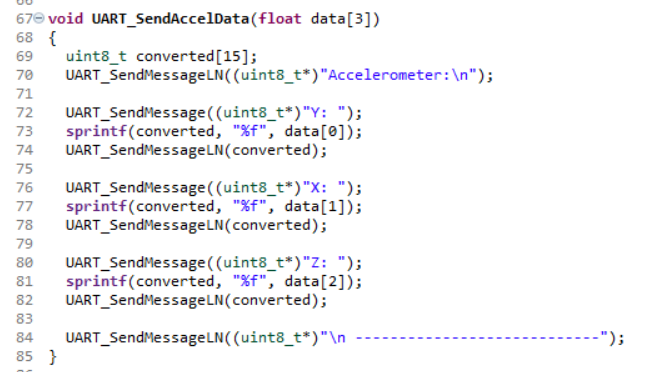
## Tests

### Raw data

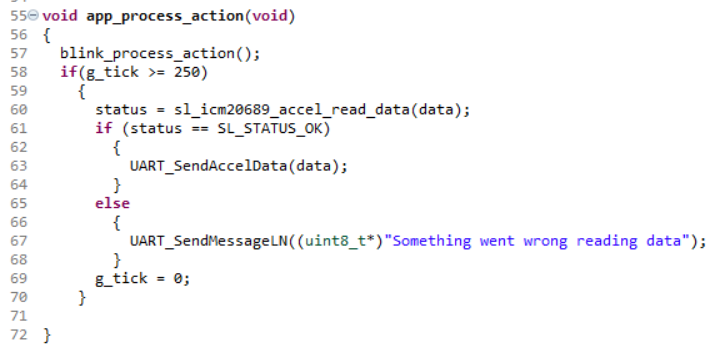
To test the motion sensor I have used a simple bare-metal project using the Gecko SDK. I added UART to send the sensor data to a terminal. To use UART I had to install the software component for this. I also created a tick timer to send the data with an interval of x ms.I created a file called message.c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



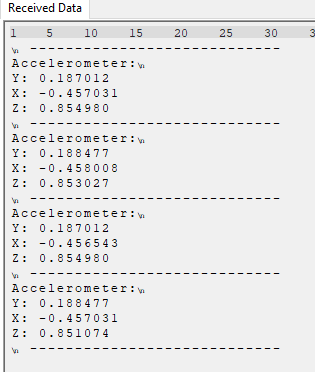
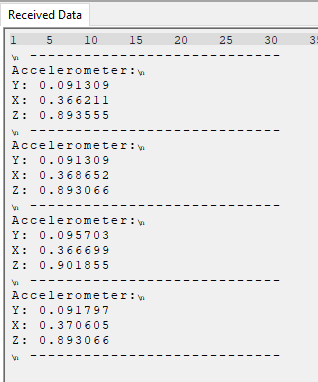
Tick\_init starts a periodic timer and the tick\_cb gets called every time the timer period times out. The UART\_SendMessageLN sends a string with linefeed ad the end while UART\_SendMessage does not. The functions UART\_SendGyroData and UART\_SendAccelData will send the data coming from the sensor. This function expects a array so we can send the 3-axis at once and uses also the SendMessage functions to send the data. It also uses sprintf to convert the float value to a character.



In the application function “app\_process\_action” the tick timer increments every 1ms the variable g\_tick and every 250 milliseconds this block of code gets executed. So every time this gets executed we read the sensor and store the data in a float array that will be send with the UART\_SendAccelData function.



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

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

### Converted data

The floating point value that the sensor gives is not usable in the Matter application. In the Matter application I use the attribute Level Control that stores a uint8 datatype. So we need to convert the floating point variable to a uint8 variable.

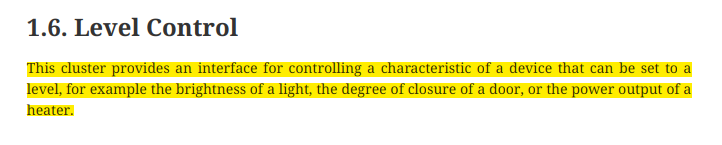
# Issues

## Gyroscope 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 a existing device type to get to our use case.

### Possible solution

* When I dived deeper in the Device Library I’ve noticed that there is actually one device type that we can use, the On/Off Sensor. The On/Off sensor contain a cluster Level control. The level control can be anything and is not something specific.



This can solve the problem on the side of the Client (the client can manipulate the server, the server only holds value’s and do something with it).

* Another solution that I found was to create a custom app, combine this with the device type and the Level control cluster. with the instructions from Silabs, It can be found in the Silicon Labs Matter Github repo.

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

First I thought that the interaction between the matter protocol and the game can be done by the read commands or to subscribing to an attribute of a endpoint (for example the on/off attribute that holds the on/off state) using the chip-tool. While doing some [tests](#_Simple_on/off_device) it seems that we get a lot of information and not only the state of a device, for example a ‘True’ value as the button is pressed. So getting that values from the chip-tool in the command line isn’t that simple. I don’t even know it is possible to filter everything so I get only the information I want. For controlling the game I need only the value of the attribute, whether it is two on/off “devices” one for left one for right or it is a level control.

### Possible solution

Maybe to make this work I can built 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 these pin’s so we can control the breakout game with this pins?

# Tests

## Simple on/off device

The first test was a simple on/off device to see what the protocol does. Before we can do any tests a matter hub is necessary. I followed the instructions from the silabs demo developer documentation. <https://docs.silabs.com/matter/1.0.1/matter-thread/demo-overview>

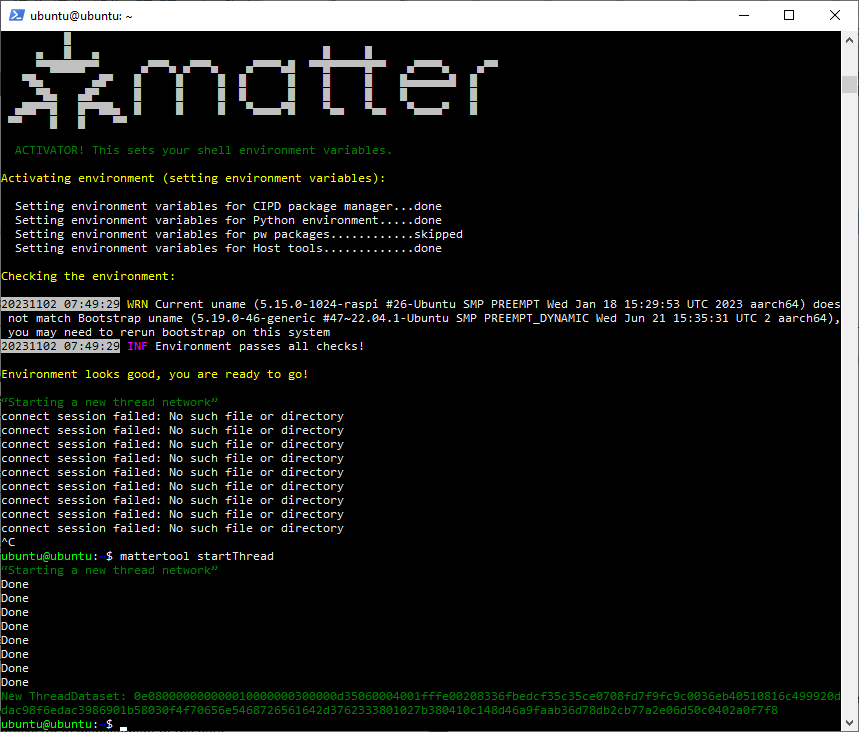
The first thing is to setup a Matter Hub. The Matter Hub consists of the Open Thread Border Router (OTBR) and the chip-tool running on a Raspberry Pi. Silicon Labs has developed a Raspberry Pi image combining the OTBR and chip-tool that can be downloaded and flashed onto an SD Card.

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 than 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.

Once the image is flashed and inserted into the RPI I searched for the IP-address. Then logged in using SSH. When logged in I had to use a few commands that are listed in <https://docs.silabs.com/matter/1.0.4/matter-thread/chip-tool>

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



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

Geef screenshot

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

Screenshot hier

So now I can control the light using this commands but I want to know if I can monitor this. So I searched in the silabs matter docs under the section “matter over Thread” and then under “Using the Chip-Tool” if there was a possibility to monitor the endpoint. What I found was that mattertool is just an alias of chip-tool. So I searched for this and found the documentation on Github: <https://github.com/project-chip/connectedhomeip/blob/master/docs/guides/chip_tool_guide.md>

In this guide there is described how to subscribe to an attribute and that was exactly what I needed. When subscribing to an attribute every change displayed in the command line. To subscribe we have to follow a few steps.

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.
2. Use the command “**<cluster-name of choice> subscribe**”. The list of all available attributes for the cluster will appears.

Geef hier een screenshot

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

**<cluster-name> subscribe <argument> <min-interval> <max-interval> <node\_id> <endpoint\_id>**

The parameters of this command:

<cluster-name> is the name of the cluster.

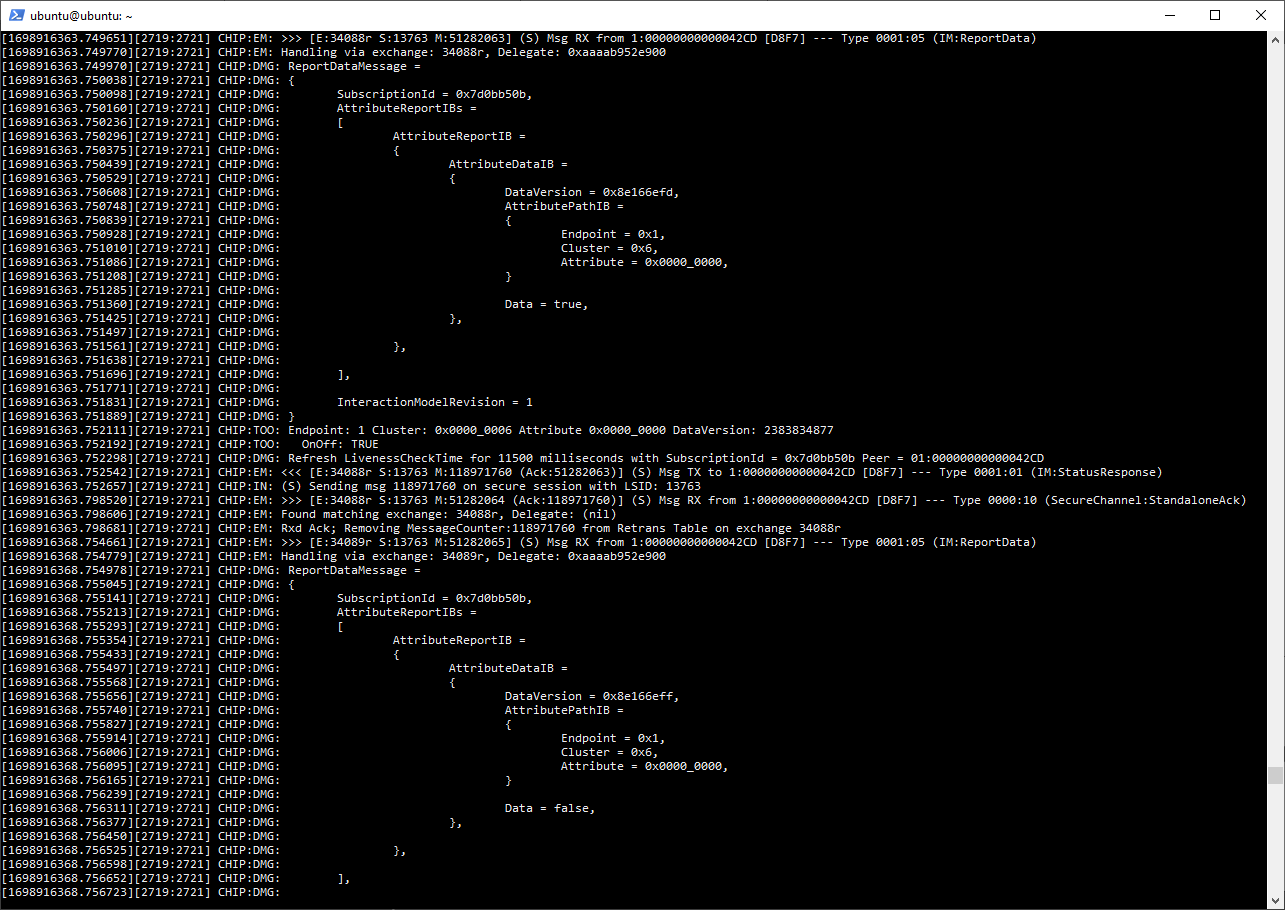
<event-name> is the name of the chosen event.

<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.

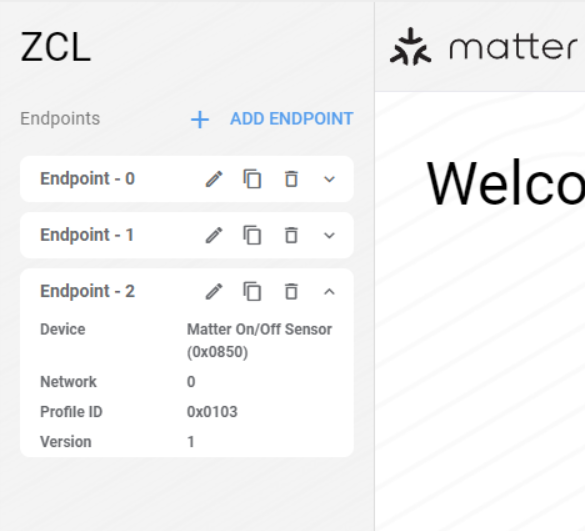
<endpoint\_id> is the ID of the endpoint where the chosen cluster is implemented.



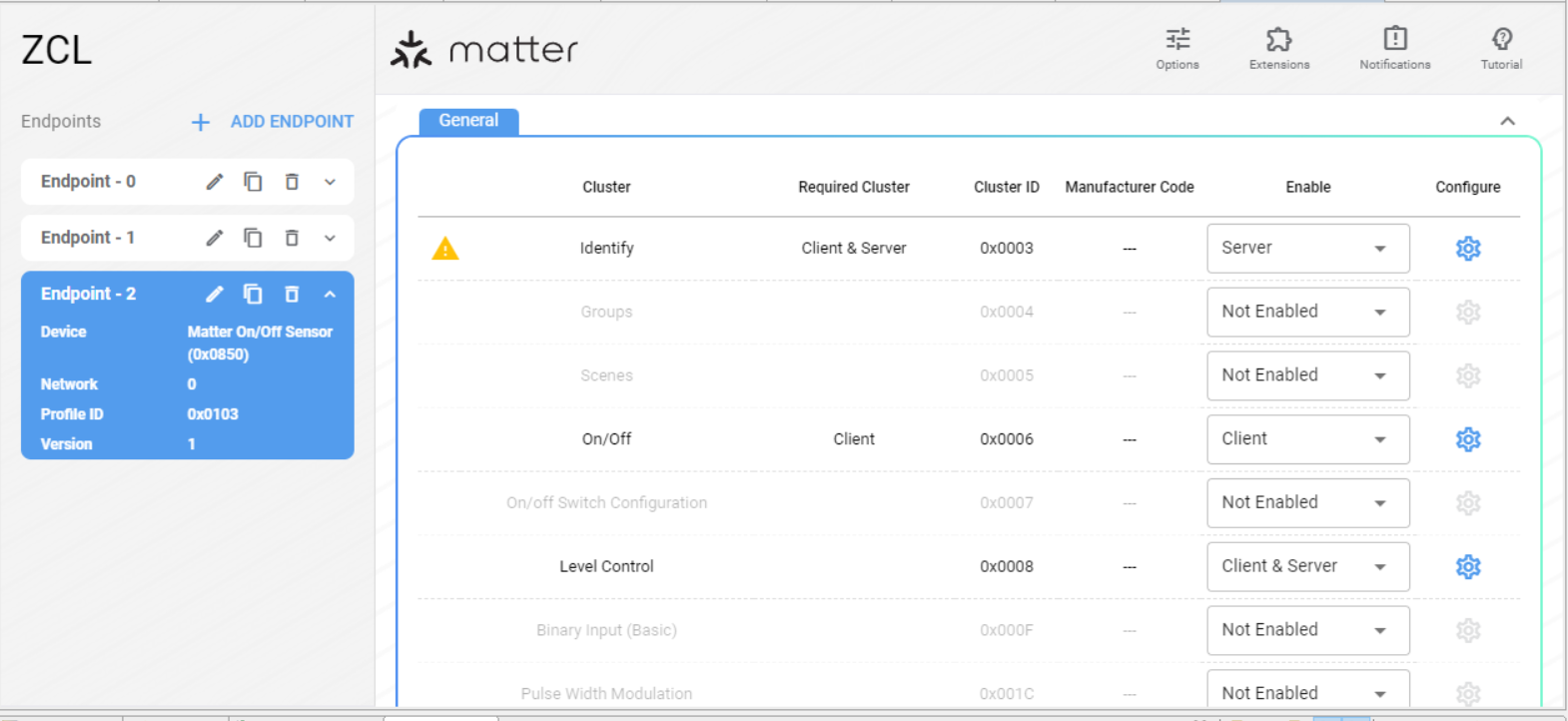
## Level control test

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

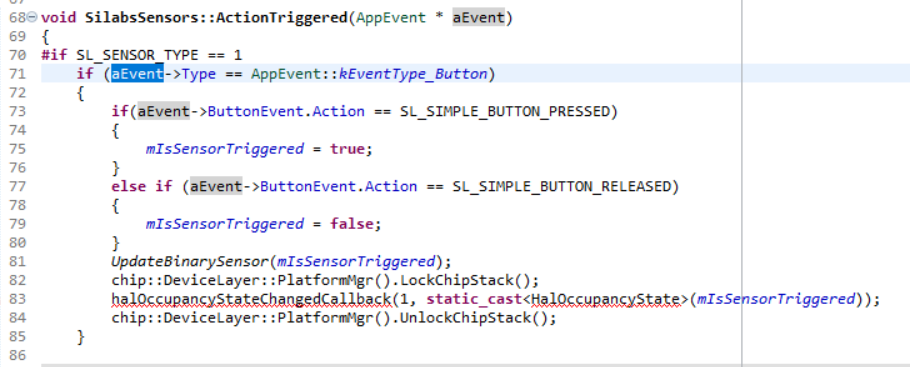
First step was the Zigbee Cluster Configuration tool in Simplicity Studio v5. I added a new endpoint with the device type “Matter On/Off Sensor”. I want to use that cluster to store the Gyroscope data in the future.



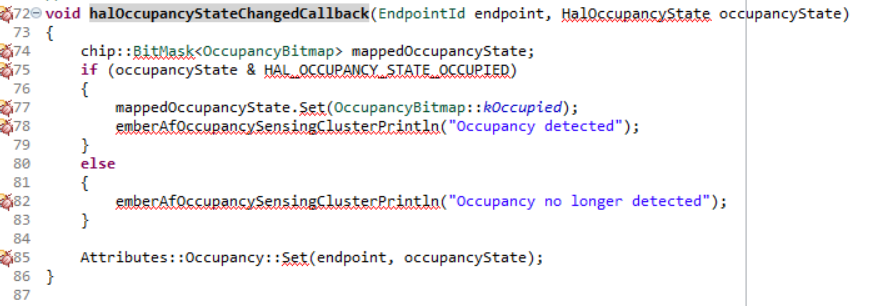
After that I enabled the level control cluster as a Client & Server in the General tab, actually we need only the Server cluster but I select both to test this.



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.



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



On line 85 they set the state by using Attributes::Occupancy::Set(). This is what we are going to need for our sensor. We are going to do the same for the level-control.

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:

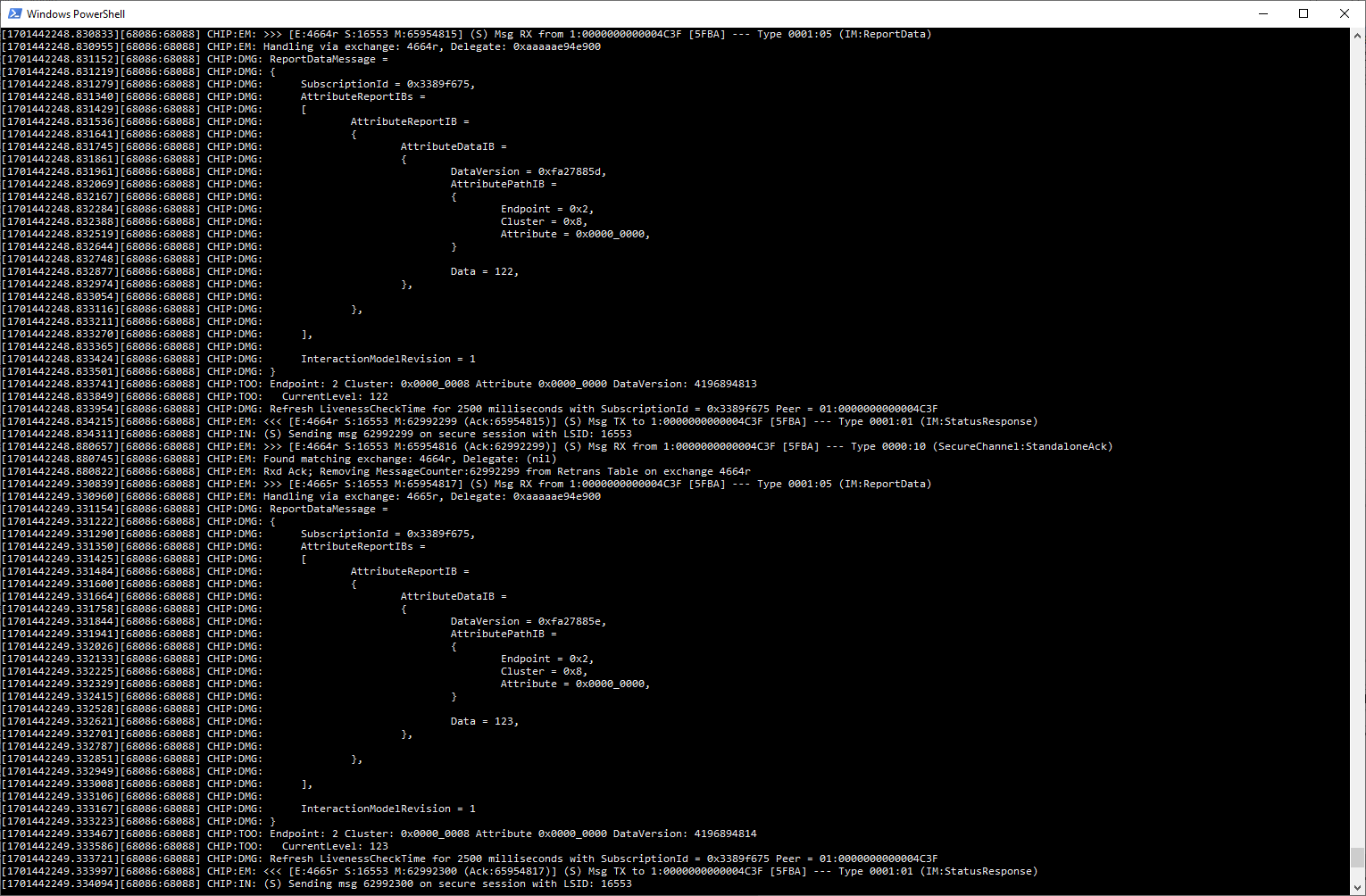
#include *<app-common/zap-generated/attributes/Accessors.h>*

Vertel over de aanpassingen in de AppTask.c

### 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. Use the following commands:

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



## Terminal output test

In the end we want to use the terminal output to control the game. In the previous tests the data is successful shown in the terminal

# Application

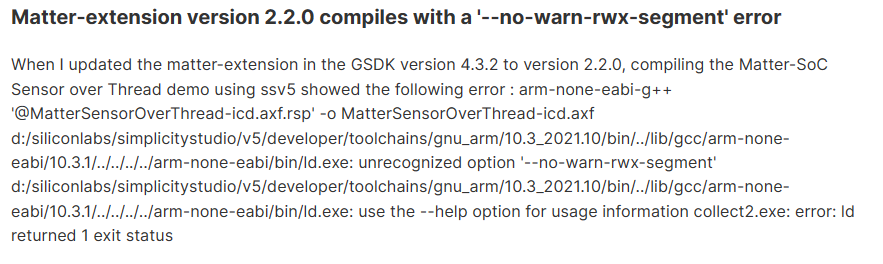
All the components are individually tested and it needs to bring together.

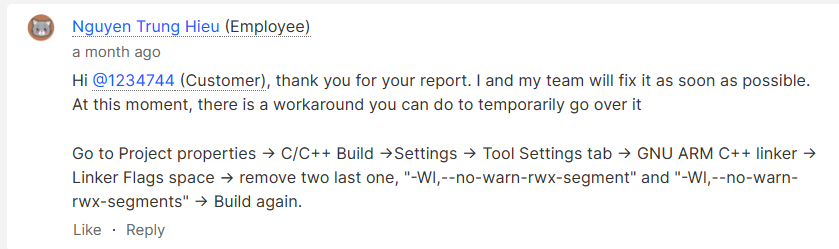
# Notes

* **READ THE DETAILED DEVELOPMENT TOPICS** [Overview Guides - latest - Silicon Labs Matter Silicon Labs (silabs.com)](https://docs.silabs.com/matter/latest/matter-overview-guides/matter-provisioning)
* **Importand to read** [Developing with Silicon Labs Matter - latest - Silicon Labs Matter Silicon Labs (silabs.com)](https://docs.silabs.com/matter/latest/matter-start/)
* **How to use the On/Off server with level control to react to the gyroscope**
  + Use a callback function when the gyro reaches a sertain degrees
  + In the callback function include the level-control.h
  + Add app-common/zap-generated/attributes/Accessors.h  
    This is necessary for using the Set function to set the value to the attribute.
  + Add using namespace chip::app::Clusters::LevelControl
  + Add the following line to set the value Attributes::LevelControl::CurrentLevel::Set(endpoint, value) OR when not adding using namespace ….. add chip::app::Clusters::OnOff::Attributes::LevelControl::CurrentLevel::Set(endpoint, value)

**Problem with SSv5**

[Matter-extension version 2.2.0 compiles with a '--no-warn-rwx-segment' error (silabs.com)](https://community.silabs.com/s/question/0D5Vm000000kGczKAE/matterextension-version-220-compiles-with-a-nowarnrwxsegment-error?language=en_US)





# Document Information

## Version History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Author** | **Version** | **Description** |
| 22/09/2023 | BGY | 0.1 | Project overview and wireless controller description |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Related Documents

|  |  |
| --- | --- |
| **Ref** | **Description** |
| 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. |