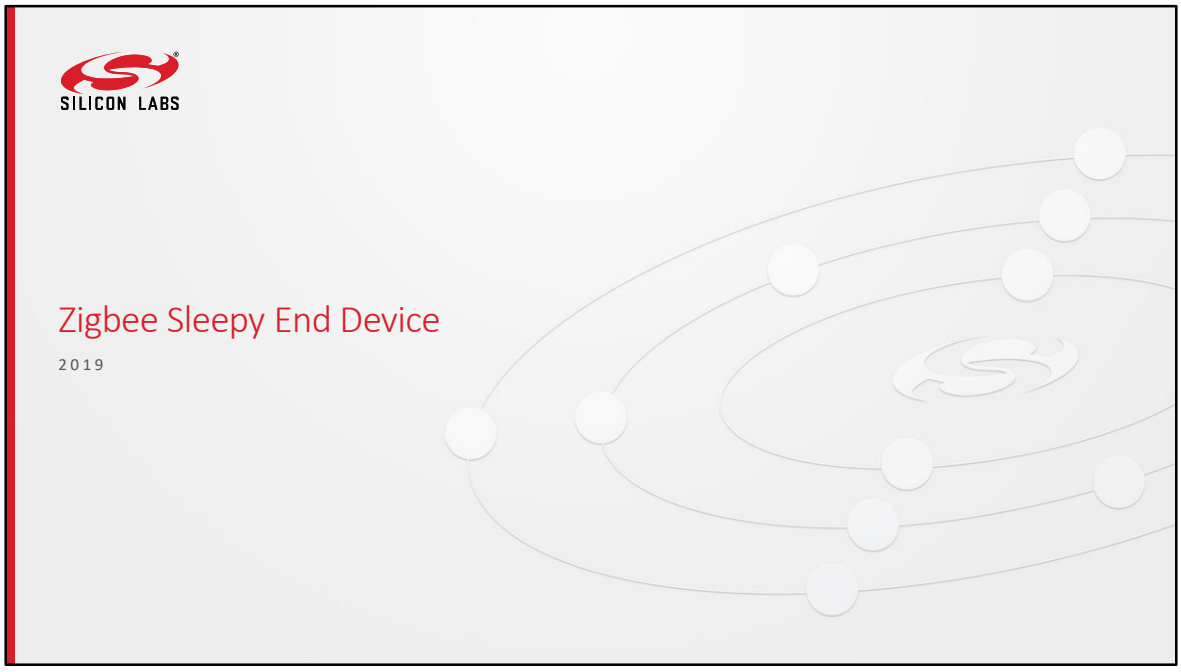


Welcome to this training. Today we will do a hands-on about sleepy end device. First we will introduce some knowledge about this hands-on.

Agenda

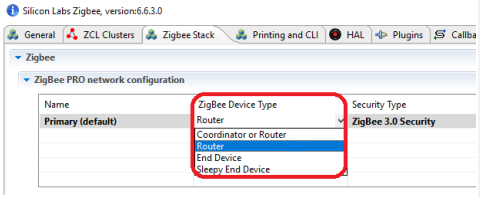
- Basic concepts of sleepy end device
  - End Device and Sleepy End Device
  - Polling
  - Timeout and Keepalive
- Step-by-Step hands-on: temperature sensor
  - Hands-on overview
  - Temperature sampling
  - Reporting
  - Finding and Binding
  - Sleep
- Q & A

2

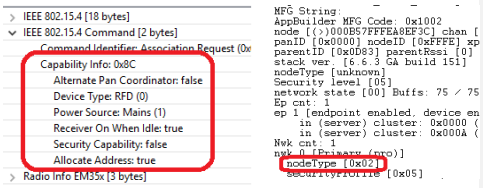
First, we will introduce some basic knowledge about sleepy end device.
Then we will introduce about the hands-on. In this hands-on, we will develop a temperature sensor, includes the temperature sampling, reporting, finding and binding, sleeping.

What are End Devices ?

- End Device (ED)
  - Nodes on Zigbee network that don't participate in routing
  - Communicate only through the parent node
  - Use data polling as keep-alive mechanism with parent
  - Rejoin to another router device when the parent is lost.
  - Able to receive messages at any time, not dependent on data polling



- Sleepy End Device (SED)
  - A special category of end device that turns off the radio when idle
  - Parent node holds messages meant for the Sleepy end device
  - Polls the parent node periodically to receive incoming messages
  - No data is sent to SED unless it is requested by said device



1 – coordinator
2 – router
3 – end device
4 – sleepy end device

- In a Zigbee network, there are three types of devices:, Coordinators, routers and end devices. This training module focus on the end devices. If you are not familiar with the node types yet, please review the "Zigbee Introduction: Node Types, PAN IDs, Addresses" training module.

- **End devices** are **leaf nodes**. They communicate only through their parent nodes and, unlike router devices, cannot relay messages intended for other nodes. They don't participate in any routing. End devices rely on their parent routers to send and receive messages. End devices that do not have tight power consumption requirements may choose to have their radio on at all times. These end devices are known as RX-on-when-idle devices.

- The **Sleepy End Device** is a special kind of end device, that turns off its radio when idle, which makes it a suitable choice for battery operated devices.

What is Polling?

- **The Main Function of Polling**
  - keep-alive message between the child and parent
  - SED may use polling to request data from parent
- **Long Poll**

The maximum amount of time an end device shall wait before polling its parent.

- **Short Poll**

The amount of time that an end device may wait before polling its parent when it is in the process of sending or receiving a message.

- **Fast Polling**

The state during which the stack actively polls its parent at the rate of short poll interval

- **Poll Control Cluster**

Providing a mechanism for the management of an end device's MAC Data Request rate

Events total:242 shown:209 Decoders: EmberZNet 6.5, ZigbeePro

Time	Type	Summary	MAC Src	MAC Dest
-0.087181	Packet	Link Status	029E	FFFF
0.000000	Packet	ZCL: Toggle	1E3B	0000
0.001904	Packet	802.15.4 Ack	0000	1E3B
1.013393	Packet	Data Request	1E3B	0000
1.014144	Packet	802.15.4 Ack	0000	1E3B
1.015078	Packet	ZCL: DefaultResponse	0000	1E3B
1.017078	Packet	802.15.4 Ack	1E3B	0000
1.031373	Packet	Data Request	1E3B	0000
1.032124	Packet	802.15.4 Ack	0000	1E3B
1.034013	Packet	APS Ack	0000	1E3B
1.035853	Packet	802.15.4 Ack	1E3B	0000
1.038535	Packet	APS Ack	1E3B	0000
1.040343	Packet	802.15.4 Ack	0000	1E3B
2.027752	Packet	Data Request	1E3B	0000
2.028504	Packet	802.15.4 Ack	0000	1E3B
3.040828	Packet	Data Request	1E3B	0000
3.041579	Packet	802.15.4 Ack	0000	1E3B
9.324688	Packet	Link Status	0000	FFFF

Short polling when sending and receiving a message

4

Now that we are familiar with end-devices, lets talk about polling.

- Polling is the event wherein an end device sends a "**data request message**" to its parent node.

- Polling has **2 main purposes** :
  - KEEP ALIVE : End devices poll their parent nodes periodically as a keep-alive mechanism to prevent being aged out of the network.
  - REQUEST MESSAGES: On the sleepy end device, polling is additionally used to request messages sent to it that are held by the parent node.

- The **Long Poll** Interval represents the **maximum amount of time between MAC Data Requests** from the end device to its parent. When the device does not need to be responsive on the network, it polls its parent on the LONG_POLL interval.

- The **Short Poll Interval :** When a device needs to be responsive to messages being sent to it from the network, it goes into a state where it polls its parent on the SHORT_POLL interval. This ensures that any messages received by its parent will

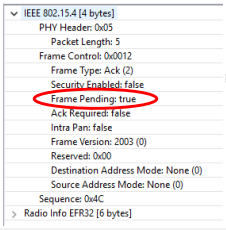
immediately be retrieved by the sleepy end device and processed. The time during which the sleepy end device is polling at the SHORT_POLL interval is referred to as "**Fast Polling mode"**. When the device expects data (such as the zcl/zdo message responses, etc.), it enters fast polling mode. Sometimes a sleepy device needs to stay in fast poll mode while sending a complex series of messages that constitute a complete application level transaction with another device. The usage of this API is documented in app/framework/include/af.h.

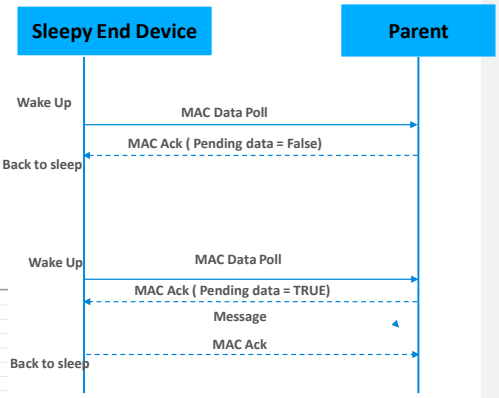
- The **packet trace** on the right was captured using the Silicon Labs Network Analyzer. It shows the end device polling its parent at the short poll interval(1 second) for 3 seconds (which is determined by Wake timeout) after sending a ZCL toggle command and expecting the default response.

- **Poll Control cluster** provides a mechanism for the management of an end device's data polling rate with ZCL command. the details of Poll Control cluster are discussed further in a separate training module (App Layer: Poll Control Cluster).

Polling as a means to request data from the parent

- Used by Sleepy End Devices
- Poll at least once within the EMBER_INDIRECT_TRANSMISSION_TIMEOUT to check for data at the parent to ensure receiving incoming messages reliably
- Some sleepy end devices(example: sensors) are not expected to receive messages asynchronously. They just need to poll within the end device poll timeout





5

- Sleepy end devices do not receive data directly from other devices on the network. Instead, they **must poll their parent for data** and receive the data from their parent. The parent acts as a surrogate for the sleepy device, staying awake and buffering messages while the child sleeps.

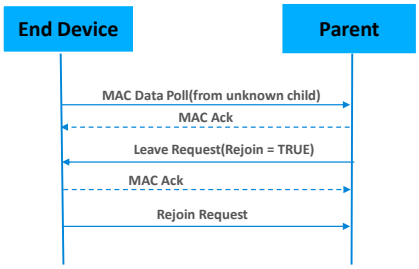
- The figure on the right illustrates the data polling process. Sleepy end devices wake up and poll their parents at regular intervals. The parent node uses the **pending data flag** in the MAC ACK to indicate that it has one or more messages waiting for the sleepy end device. If the pending data flag is true, the sleepy end device stays awake to receive the message(s) and acks them before going to sleep. If it is false, the sleepy end device is free to go back to sleep until the next poll attempt.

- Please keep in mind that if you want the device to receive incoming messages and incoming APS ACKs (for its outgoing messages) reliably, you should poll at least once within the **EMBER_INDIRECT_TRANSMISSION_TIMEOUT** (7.68 seconds by default) to check for data at the parent because the length of time that the parent will hold on to a message is determined by this number. Some sleepy end devices(such as **sensors**) are **not expected to asynchronously receive messages**, so they don't have the above limitation. They just need to poll within the **end device poll timeout**

which we will talk about in the next slide.

Keepalive and timeout

- All end devices MUST keepalive with their parents.
- End device will move to a new parent if it loses the connection of the current parent .
- Parent will remove the child when the child is not active.
- Two keepalive mechanism
  - By polling
  - By timeout request/response message



- **When is a Rejoin initiated?**
  - Parent initiates rejoin if it receives a mac data poll after the end device has been aged out of its child table
  - End device initiates a rejoin if its data polling message isn't acknowledged by the parent over EMBER_AF_PLUGIN_END_DEVICE_SUPPORT_MAX _MISSED_POLLS tries.

6

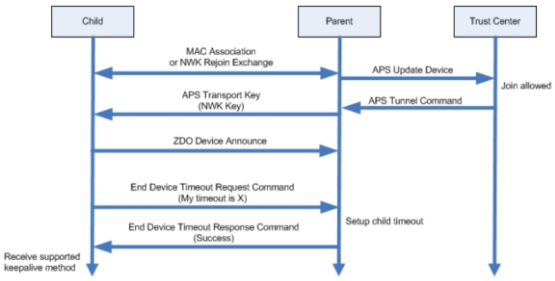
Mac Data Polling is used as **the keep-alive message** between the child and parent.

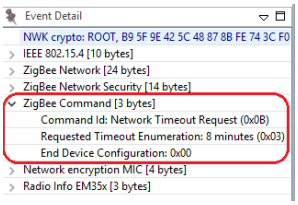
End devices have to poll their parent at least once within the End Device Poll Timeout (as set by **EMBER_END_DEVICE_POLL_TIMEOUT** or **EZSP_CONFIG_END_DEVICE_POLL_TIMEOUT**). Otherwise, these devices will be removed from the child table of the parent, effectively being aged out of the network. This is done to ensure that the child table slot is not permanently reserved for an end device that has been removed from the network ungracefully (i.e., if no Leave notification was heard from that device. Leave notifications are broadcasts and are not guaranteed to be received);

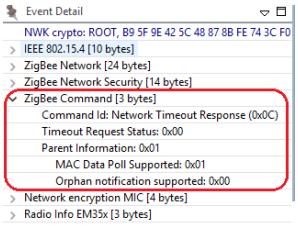
The parent node will ask the end device to **leave and rejoin** the network if it receives a mac data poll from the end device which doesn't exist in the child table. The figure on the right illustrates the leave and rejoin process.
In addition to this, if the **data polling message isn't acknowledged** by the parent for EMBER_AF_PLUGIN_END_DEVICE_SUPPORT_MAX_MISSED_POLLS times, the end device will attempt to rejoin the network to find a new parent.

End Device Poll Timeout Negotiation

- **End Device Timeout Request Command**







7

Zigbee R21 Specification offers an **end device timeout negotiation protocol** and a standard way to implement child aging. The End Device Timeout Request command is sent by an end device to inform the parent of its timeout value when joining/rejoining the network. When the parent receives this command, it will update the End Device Timeout value for this end device locally and generate an end Device timeout response command with a status of SUCCESS. Pre-R21 devices do not support end device timeout command. So they can only use the default end device timeout value set initially on the parent node.

Hands-on: Overview

- **What's the requirement for a sensor?**

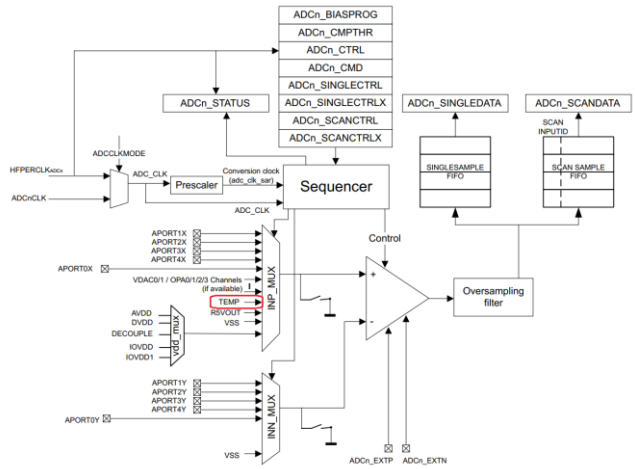

Temperature

- **Data sampling -- ADC**
- **Reporting**
  - Periodically
  - Report on change
- **Binding**
- **Sleep**

8

In this hands-on, we will develop a sleepy temperature sensor which will report temperature to the gateway periodically.
To achieve this, we need to implement the data sampling, reporting, binding and sleeping.

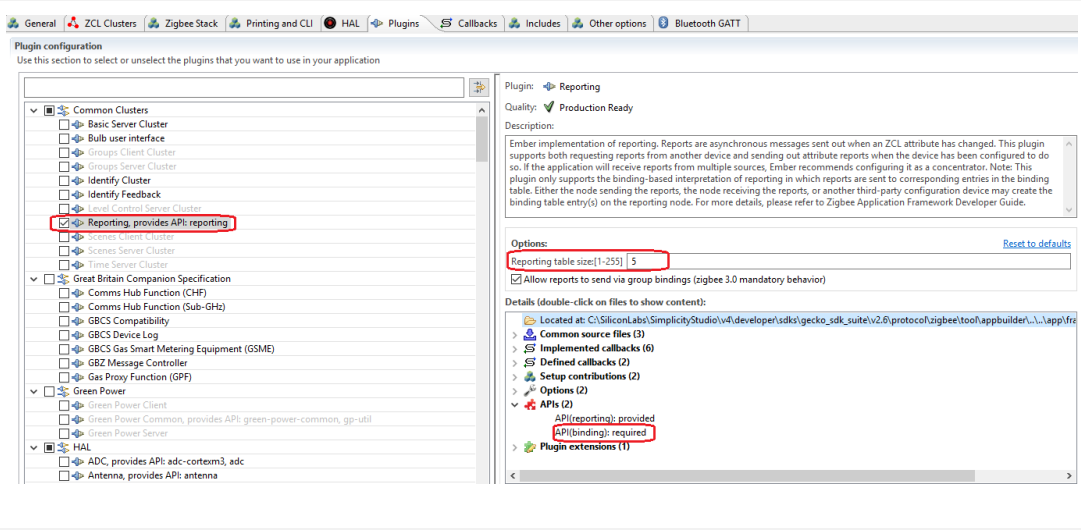
Hands-on: Data Sampling



- EFR32xG12 Reference Manual
- AN0021
- Example:
  - app\mcu_example\SLSTK3402A_EFM32PG12\inttemp

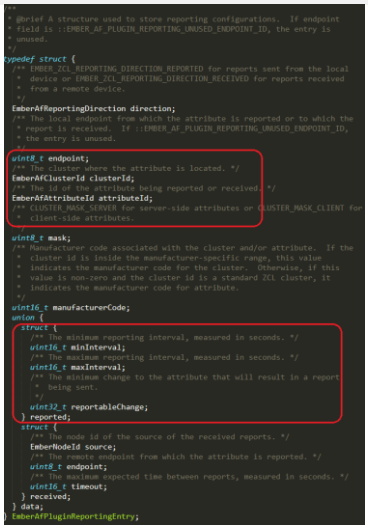
We will use the internal temperature sensor of the internal ADC to get the temperature.
Here is an example in the SDK. We can use that source code directly.

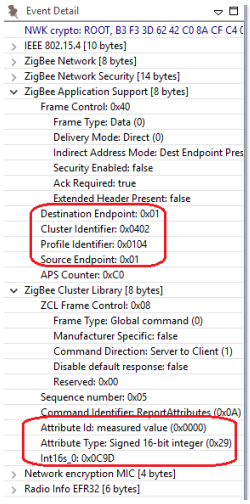
Hands-on: Reporting



To report the temperature data, we need to use the reporting plugin.
In this plugin, we need to save the attributes which will be reported into a reporting table. The table size is configurable.

Hands-on: Reporting

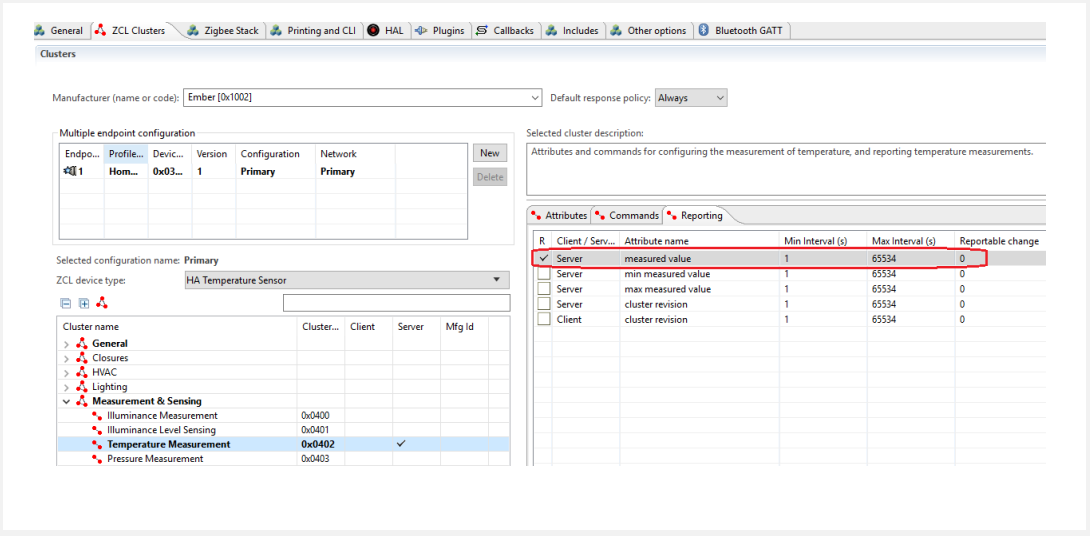




11

In the reporting table, the cluster and attribute ID will be saved. Also the interval of the reporting will be recorded in this table entry.

Hands-on: Reporting



To configure the attributes which will need to be reported, you just need to enable the reporting option of the selected attribute.

Overview of Finding & Binding

- Finding & Binding
  - Is a cluster commissioning method to establish application connection automatically
- How to start
  - Invoked by user interaction on two or more devices
- How to implement
  - Identifies and discovers end points using the Identify cluster
- What's the result
  - Binding is created at the initiator

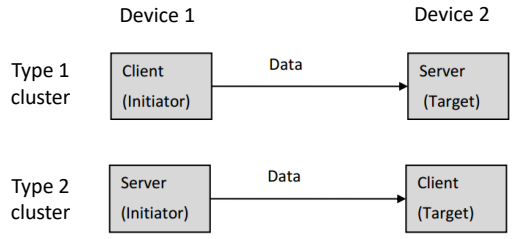
As we have set the reporting table, where will the attributes be reported?

Report message will be sent through binding tables.
There could be multiple destination of a reporting.

The binding table is set up through the finding and binding procedure per Zigbee BDB spec.

Which device needs perform Finding & Binding

- The application SHALL perform finding & binding as an **initiator** endpoint
  - Type 1 client cluster
  - Type 2 server cluster
- The application SHALL perform finding & binding as a **target** endpoint
  - Type 1 server cluster
  - Type 2 client cluster



14

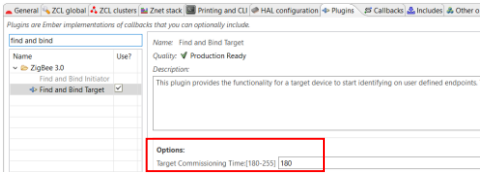
As we know, there are two kinds of clusters: Type 1 and Type 2.
A application cluster is either a Type 1 or Type 2 cluster, depends on its primary functional transactions. A transaction has an initiator and a target.

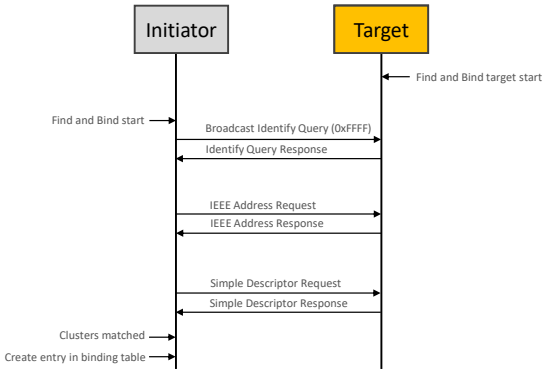
A type 1 cluster's primary function is to initiate transactions from the client to the server. For example: An On/Off client sends commands (data) to the On/Off server.
A type 2 cluster's primary function is to initiate transactions from the server to the client. For example: An Temperature Measurement server reports to the Temperature Measurement client.

For a type 1 client or a type 2 server cluster, the application shall perform finding & binding as an initiator endpoint.
For a type 1 server or type 2 client cluster, the application shall perform finding & binding as a target endpoint.

Finding & Binding procedure for a target endpoint

- Finding & Binding target
  - Write identify time attribute
  - Handle identify query requests from the initiator
- Plugin in Appbuilder





15

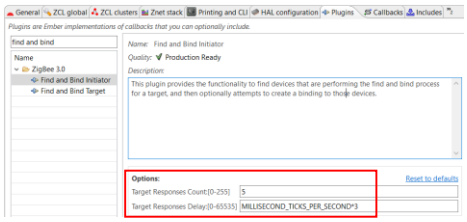
Please look at the right graph, which shows the working flow of the finding & binding procedure. We will talk about the target side firstly.

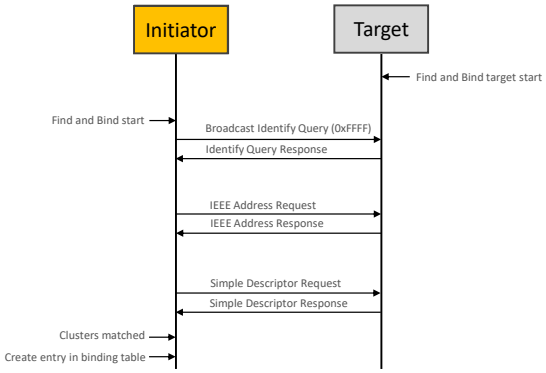
On the finding & binding target endpoint, once the finding & binding target start, it will write the identify time attribute firstly to make sure the target can be identified. In Appbuilder, the identify time can be configurated in “Find and Bind Target” plugin and the default value is 180 seconds. During the identify time, the target should respond to the identify query from initiator. Once the decrementing identify time attribute reaches zero, the target shall terminate the finding & binding procedure.

It means that the finding & binding target should write identify time attribute to make sure it can be identified during the finding & binding procedure.

Finding & binding procedure for an initiator endpoint

- Finding & Binding initiator
  - Broadcast identify query
  - Identify query responses received
  - Send simple descriptor request
  - Binding table is created for matched clusters
- Plugin in Appbuilder



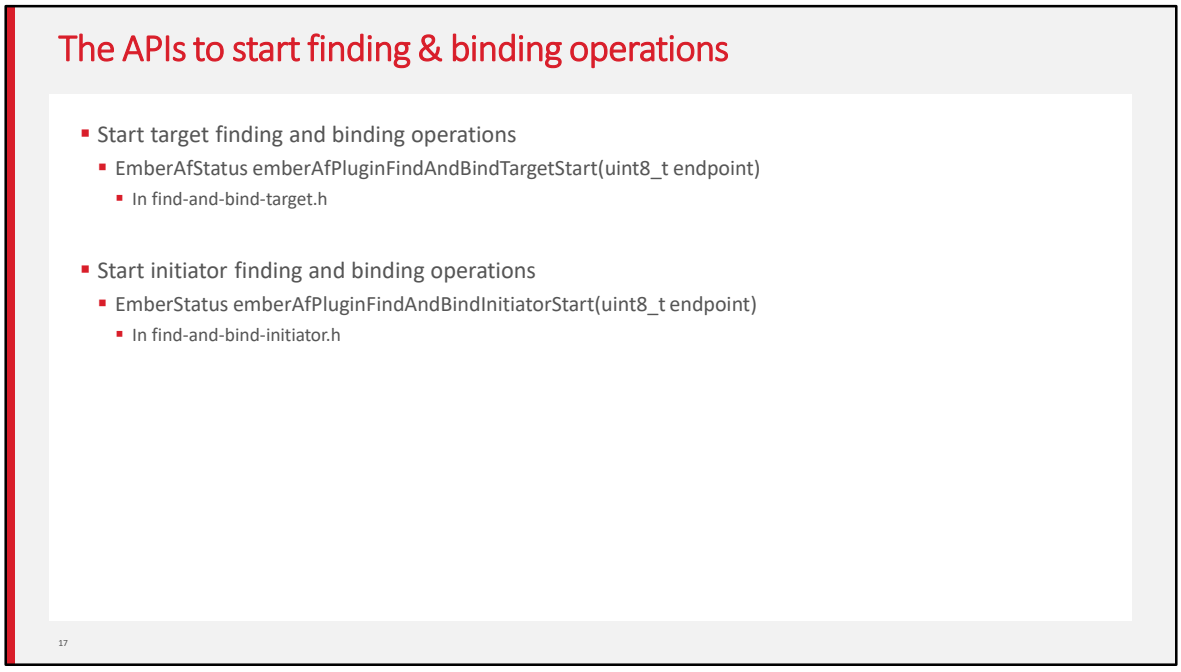


16

On the finding & binding Initiator endpoint, it broadcasts identify query to all nodes which include sleepy end device(using the broadcast address 0xffff). If no identify query response commands received, the initiator sets status to NO_IDENTIFY_QUERY_RESPONSE and terminates the finding & binding procedure.

If at least one identify query response is received, the initiator sends IEEE address request to get the EUI64 of the target, which will be used for binding table entry later. And then the initiator sends simple descriptor request to get the clusters info on target. Once some clusters matched between initiator and target endpoints, then the binding is created for every matched clusters on initiator. If a group binding is requested, the initiator endpoint configures group membership of the target endpoint, which means that the initiator unicasts “add group” command to the target.

In Appbuilder, there are two options can be configured in “Find and Bind Initiator” plugin. The “Target Responses Count” means the number of the target responses that the initiator will accept. The “Target Responses Delay” means how long the initiator will listen for target responses. You can feel free to configure these options to fit your user case.



You may want to know what are the APIs to start finding & binding operations.

The API to start target finding and binding operations is emberAfPluginFindAndBindTargetStart(), which can be found in find-and-bind-target.h. It is a call to this function will commence the target finding and binding operations. Specifically, the target will attempt to start identifying on the endpoint that is passed as a parameter. The EmberAfStatus value describing the success of the commencement of the target operations.

As the similar, the API to start initiator finding and binding operations is emberAfPluginFindAndBindInitiatorStart(), which can be found in find-and-bind-initiator.h. It is a call to this function will commence the initiator finding and binding operations. Specifically, the initiator will attempt to start searching for potential bindings that can be made with identifying targets. The EmberStatus value describing the success of the commencement of the initiator operations.

Please note that, the target should be started first during the finding & binding procedure.

Debug Commands

- Finding & Binding target

```
ZCO>plugin find-and-bind target 1
Find and Bind Target: Start target: 0x00
T00000000:RX len 3, ep FF, clus 0x0003 (Identify) FC 01 seq 00 cmd 01 payload[]
T00000000:RX len 5, ep 01, clus 0x0003 (Identify) FC 00 seq 00 cmd 0B payload[00 00]
```

- Finding & Binding initiator

```
ZSED>plugin find-and-bind initiator 1
Processing message: len=3 profile=0104 cluster=0003
T00000000:RX len 3, ep FF, clus 0x0003 (Identify) FC 01 seq 00 cmd 01 payload[]
Processing message: len=5 profile=0104 cluster=0003
T00000000:RX len 5, ep 01, clus 0x0003 (Identify) FC 09 seq 00 cmd 00 payload[72 00 ]
T00000000:TX (resp) Ucast 0x00 TX buffer: [00 00 0B 00 00 ]
Processing message: len=12 profile=0000 cluster=8001
Processing message: len=25 profile=0000 cluster=8004
Find and bind initiator complete: 0x00
```

- Binding Table info

```
ZSED>option binding-table print
#  type    nwk  loc   rem   clus   node   eui
0: UNICA   0    0x01  0x01  0x0003 0xD0E4 (>)000B57FFFE648DA0
1: UNICA   0    0x01  0x01  0x0006 0xD0E4 (>)000B57FFFE648DA0
2: UNICA   0    0x01  0x01  0x0008 0xD0E4 (>)000B57FFFE648DA0
3 of 10 bindings used
```

Transactions total:50 shown:50

Time	Duration	Summary	NWK Src	NWK Dest	P#	M#	E#	Status
10:13:41.179	0.948	Many-to-One Route Discovery	0000	FFFC	7			
10:13:47.586	0.030	ZCL: IdentifyQuery	3C88	FFFF	3			
10:13:47.599	0.030	ZCL: IdentifyQueryResponse	D0E4	3C88	2			
10:13:47.623	0.013	ZCL: DefaultResponse	3C88	D0E4	2			
10:13:50.612	0.011	IEEE Address Request	3C88	D0E4	2			
10:13:50.618	0.008	Ieee Address Response	D0E4	3C88	2			
10:13:50.628	0.012	Simple Description Request	3C88	D0E4	2			
10:13:50.634	0.013	Simple Description Response	D0E4	3C88	2			
10:14:43.402	1.115	Many-to-One Route Discovery	0000	FFFC	7			
10:15:45.618	0.942	Many-to-One Route Discovery	0000	FFFC	7			

Packet trace of Finding & Binding procedure

18

It is easy to set up the testing for finding & binding with Z3LightSoc and Z3SwitchSoc samples.
First you can build the two samples directly, download the firmware to the kits separately, and then join to the same Z3.0 network formed by a Z3Gateway.
On the Z3LightSoc side, launch the console and type CLI command "`plugin find-and-bind target 1`".
On the Z3SwitchSoc side, launch the console and type CLI command "`plugin find-and-bind initiator 1`".

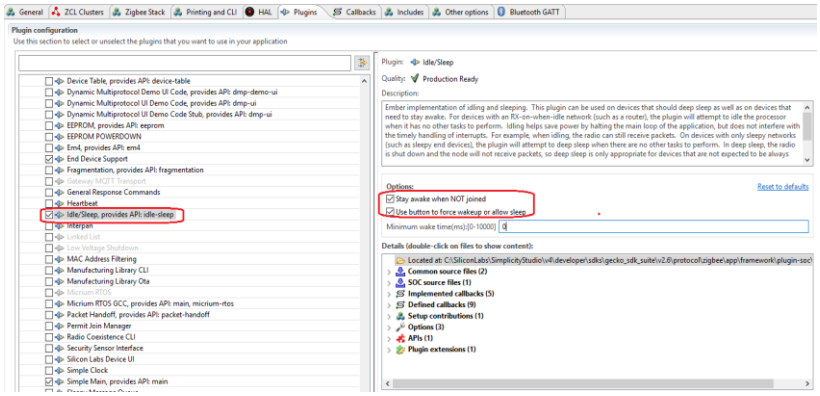
You will see the log "Find and bind initiator complete: 0x00" is printed on console after the finding & binding procedure finish. When you print the binding table on Z3SwitchSoc side(initiator), you will see the entries are created in binding table. All the matched clusters between initiator and target are bound. The finding & binding transactions can be found in packet trace, which proves the finding & binding working flow works as expected.

Let's summarize how the finding and binding procedure works.

On the target side, it will write the identify time attribute for

EMBER_AF_PLUGIN_FIND_AND_BIND_TARGET_COMMISSIONING_TIME. During the duration, the initiator broadcasts identify query and the target responds identify query response. Then the initiator sends IEEE address request to target to get the EUI64 of target, which will be used in creating binding table. The simple descriptor request will be sent to target to get the clusters info on target side. Once the clusters matched, the entries will be created in binding table of initiator.

Sleeping



- Press button0 to force device to stay awake
- Press button1 to allow device to sleep

19

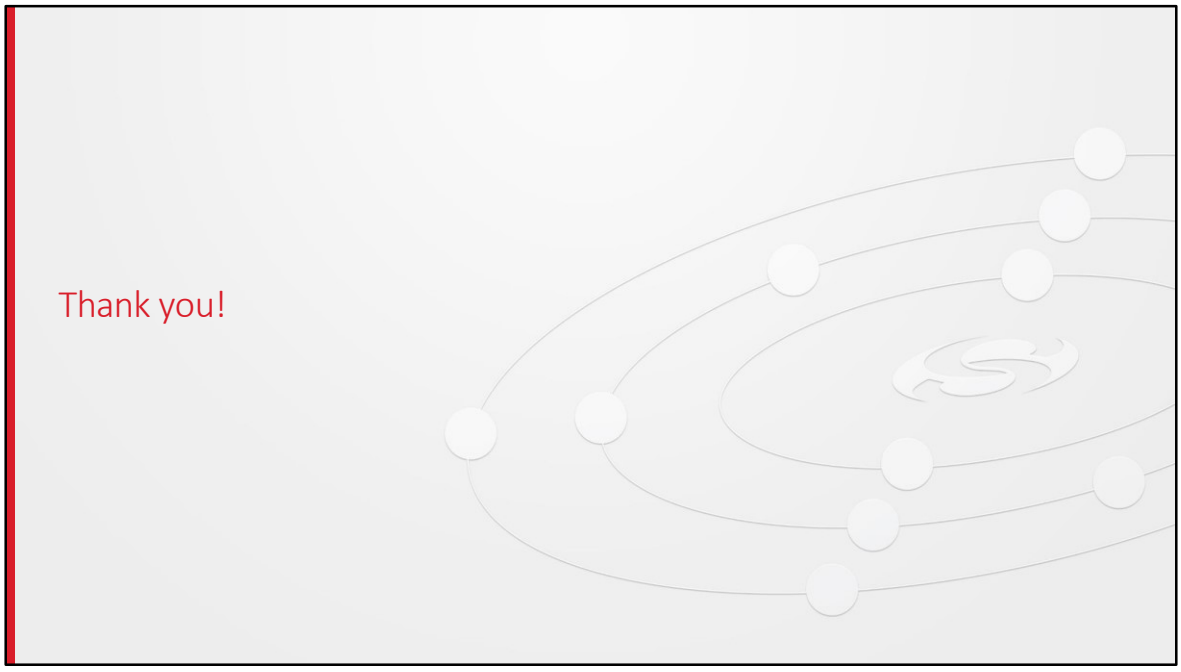
To make the device sleep, we need to use the plugin “idle/sleep”.

There are two options in this plugin, we can enable the two options so that we can debug the application easily.
Press button-0 to force the device to stay awake, so that we can input debug commands.
Press button-1 to allow device to sleep, so that we can measure the sleep current.

Q&A

Q&A

Any questions?



Thanks