



ZigBee Sleepy End Device Lab Worksheet

In this worksheet we provide a step-by-step guide to create, build and run ZigBee 3.0 end device and sleepy end device applications based on EmberZNet Stack 6.6.4. If you use a later release in the future, most of the instructions should still apply, although there could be minor differences not foreseen at the time of this document.

These exercises help you get familiar with ZigBee 3.0 in the EmberZNet Stack, Simplicity Studio v4 development environment, and the Wireless Start Kit (WSTK) with EFR32MG modules. We assume that you have a WSTK and the following software requirements:

- Simplicity Studio 4
- EmberZNet 6.6.4
- GCC 7.2

KEY FEATURES

- Step-by-step guide to creating, building and running ZigBee 3.0 applications based on EmberZNet 6.6.4
- Use Simplicity Studio v4 as the development tool
- ZigBee end device polling
- Zigbee end device keepalive and aging
- Zigbee end device rejoin

1 Pre-requisites

Make sure you have installed the EmberZnet 6.6.4 SDK and GCC toolchain on your PC.

1.1 Check EmberZnet SDK

1. Launch Simplicity Studio v4.
2. “Windows”→”Preference”→”Simplicity Studio”→”SDKs”, make sure “EmberZnet 6.6.4” is installed

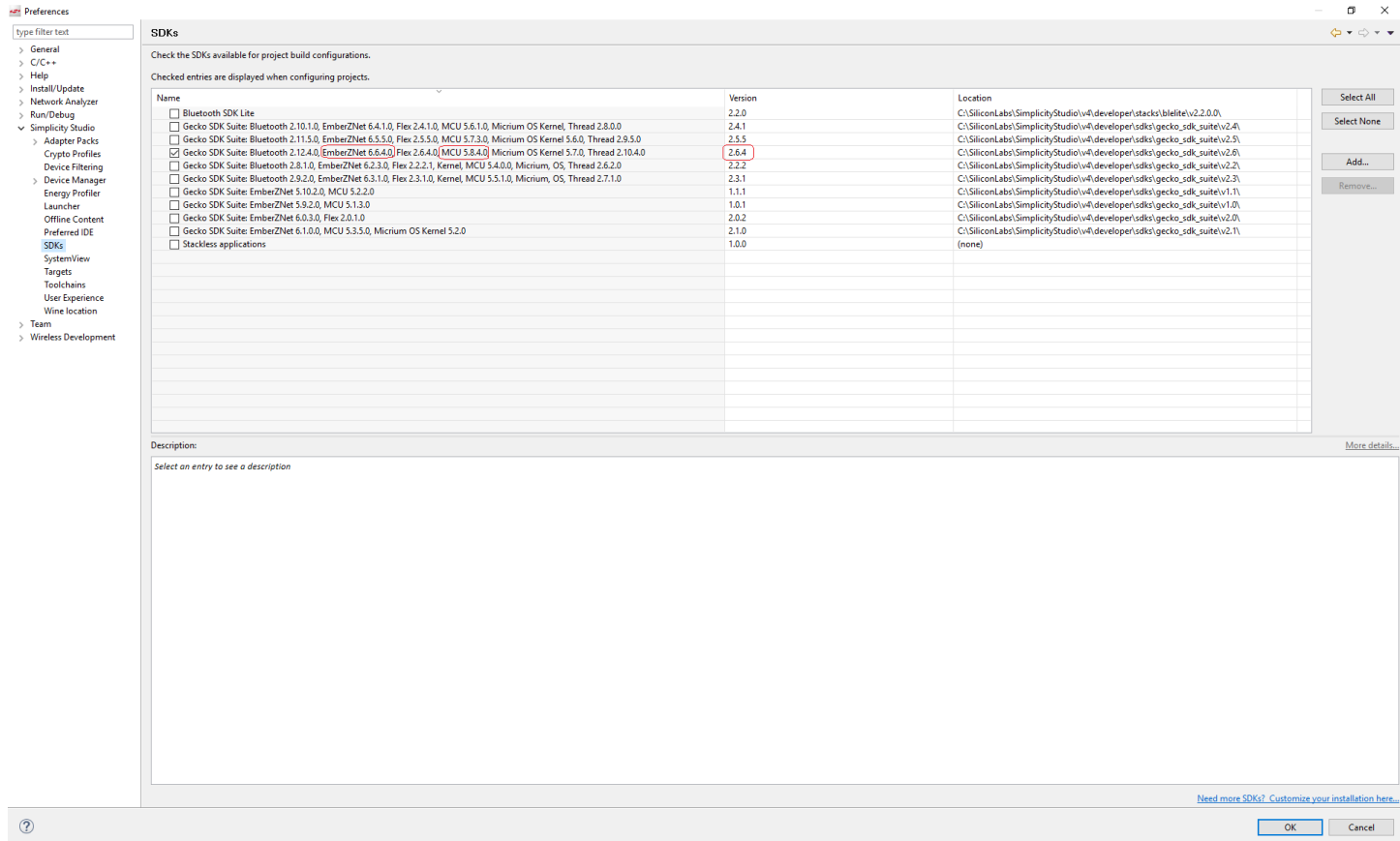


Figure 1 Check SDK in Simplicity Studio

1.2 Check Toolchains

1. Launch Simplicity Studio v4.
2. “Windows”→”Preference”→”Simplicity Studio”→”Toolchains”, make sure GCC toolchain is installed.

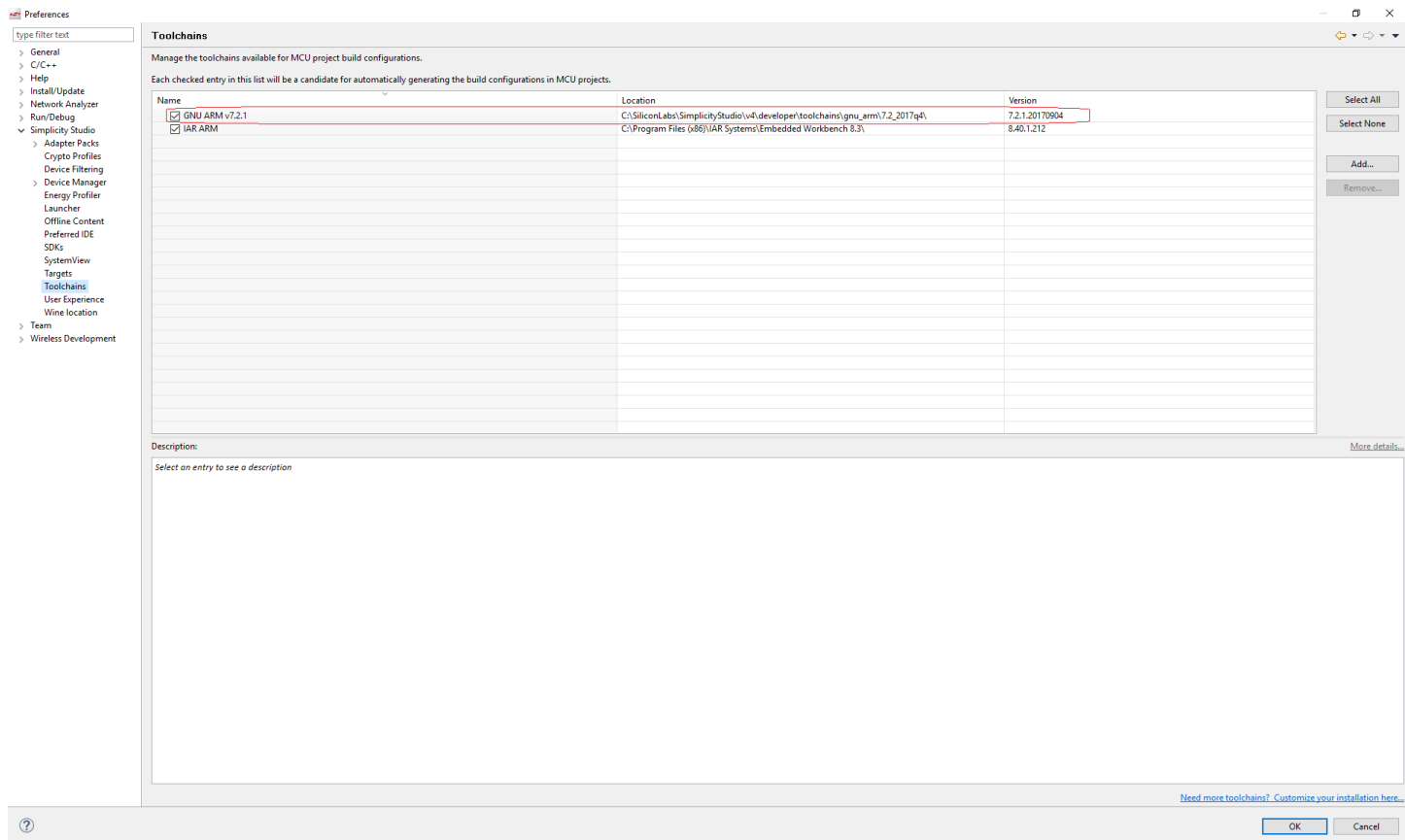



Figure 2 Check Toolchain in Simplicity Studio

1.3 Coordinator

We will provide a coordinator here and all trainee's devices can join this coordinator.

2 Flash the program

1. Start Simplicity Studio, then connect your device to PC;
2. In the menu bar, find the icon  for "Flash Programmer", press it;
3. In the popup window, select the device;

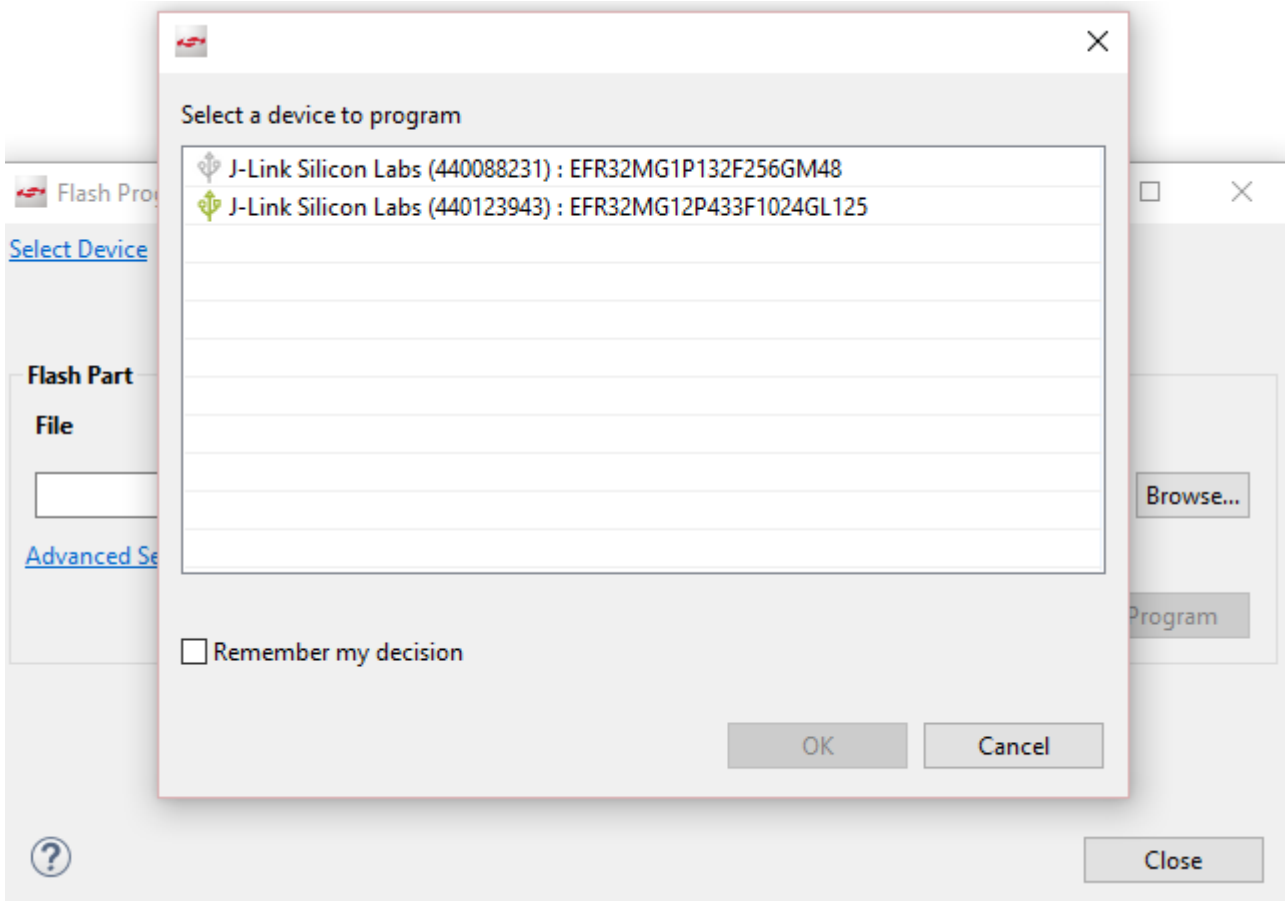


Figure 3 Select device

4. Then in the next window, click "browse" to select the image(.s37 or .hex), press "Program". You can also press "erase" if needed.

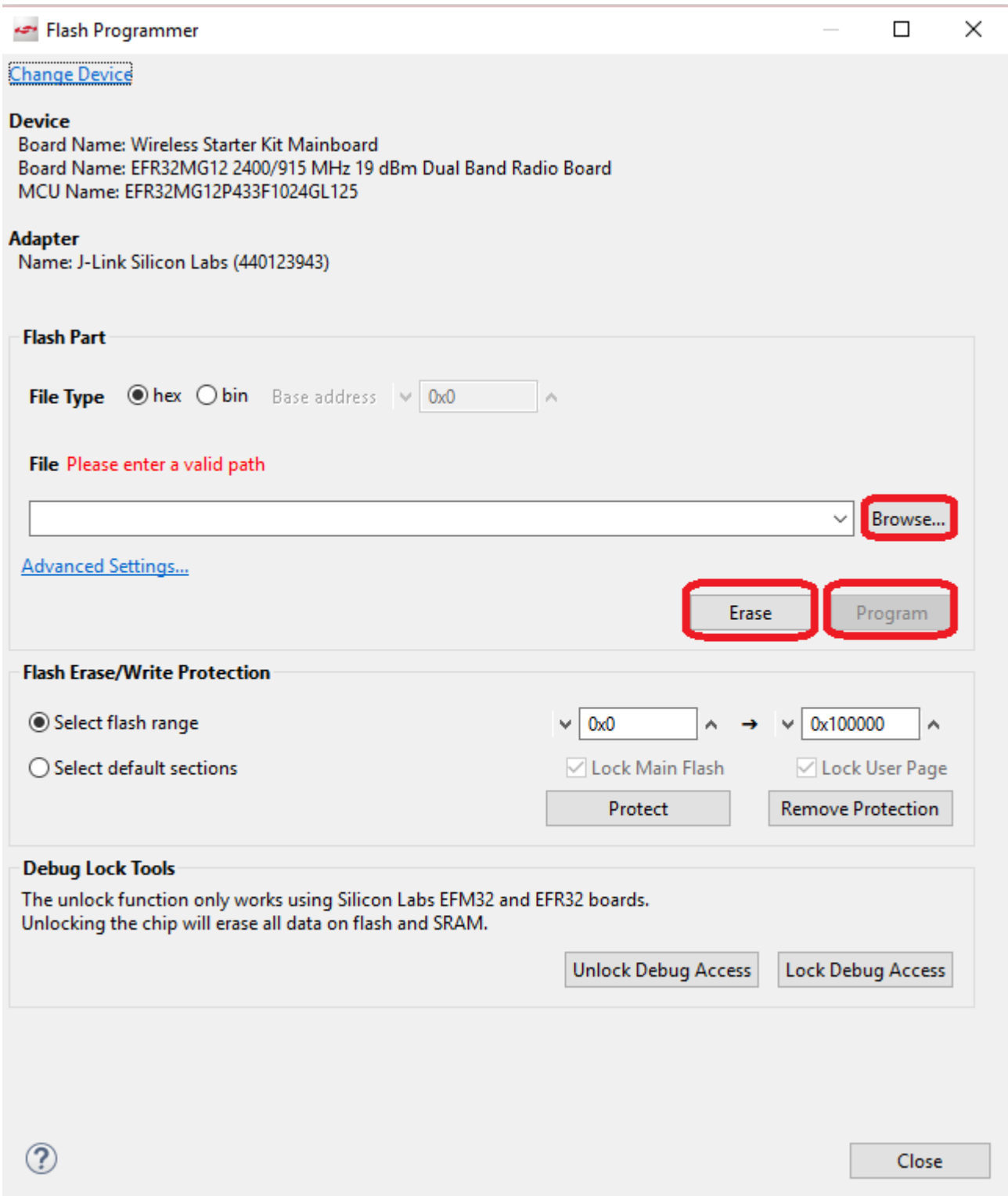
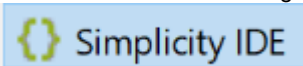


Figure 4 Flash application

3 Open console

Simplicity Studio has integrated a console so that it's convenient to debug through console. To use the console, you need:

1. Change to "Simplicity IDE" perspective:



2. Select your adapter in the “Debug Adapters” window, right click and select “connect”;
3. Select your adapter in the “Debug Adapters” window, right click and select “Launch console”;

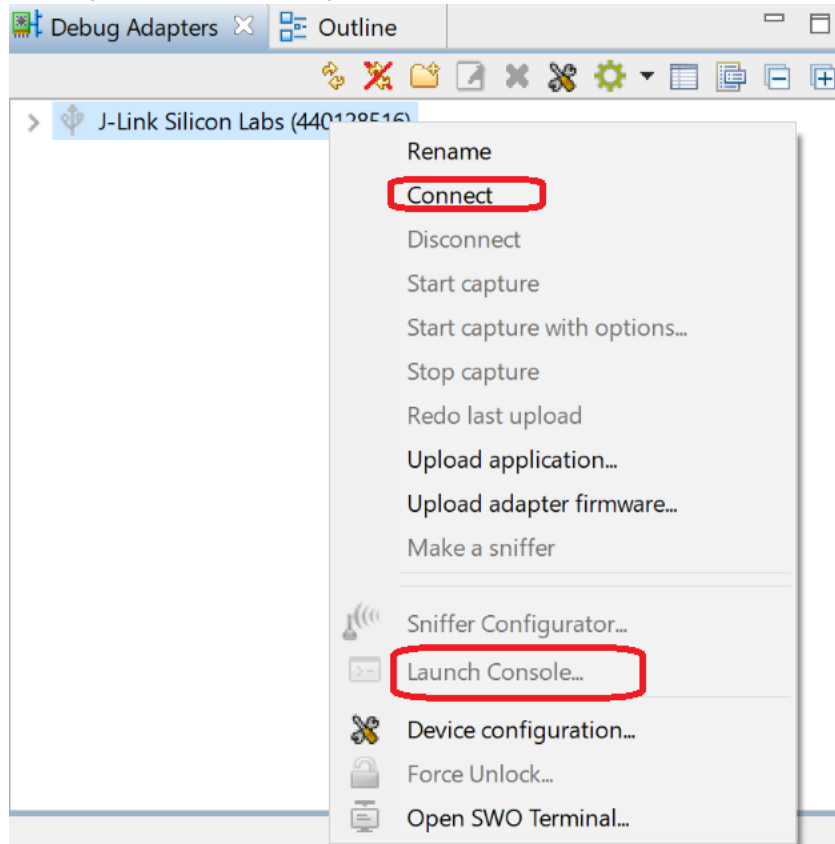
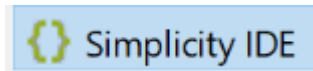


Figure 5 Launch console

4 Start Capture



1. Change to “Simplicity IDE” perspective :
2. Select your adapter in the “Debug Adapters” window, right click and select “connect”;
3. Select your adapter in the “Debug Adapters” window, right click and select “Start Capture”;

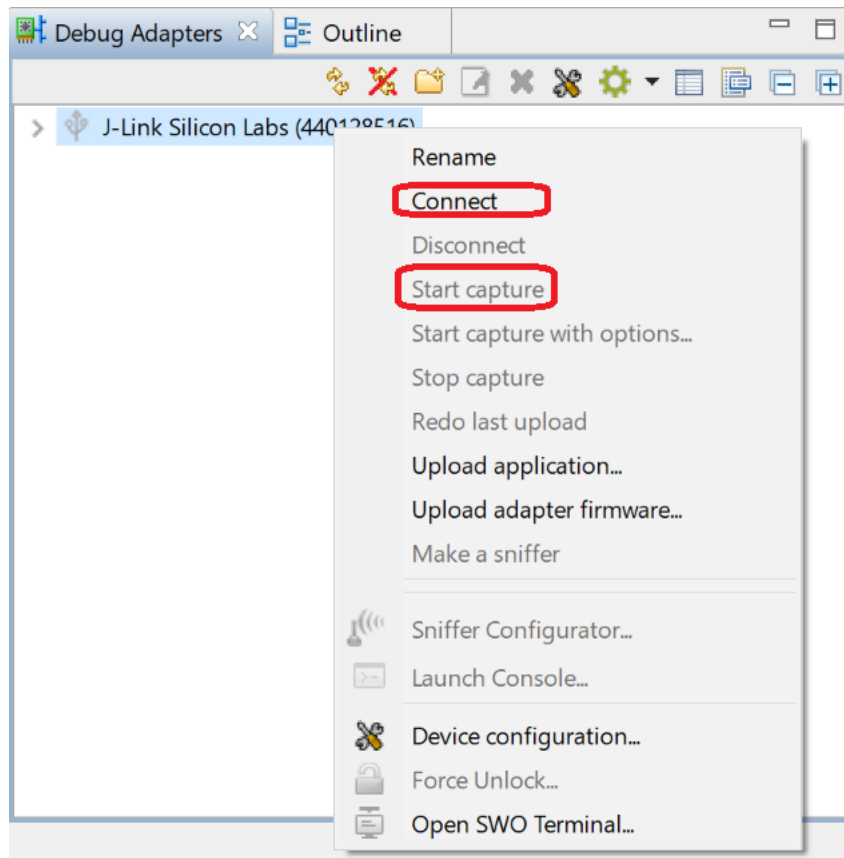


Figure 6 Start Capture

5 Start Energy Profiler

1. Start Energy Profiler :

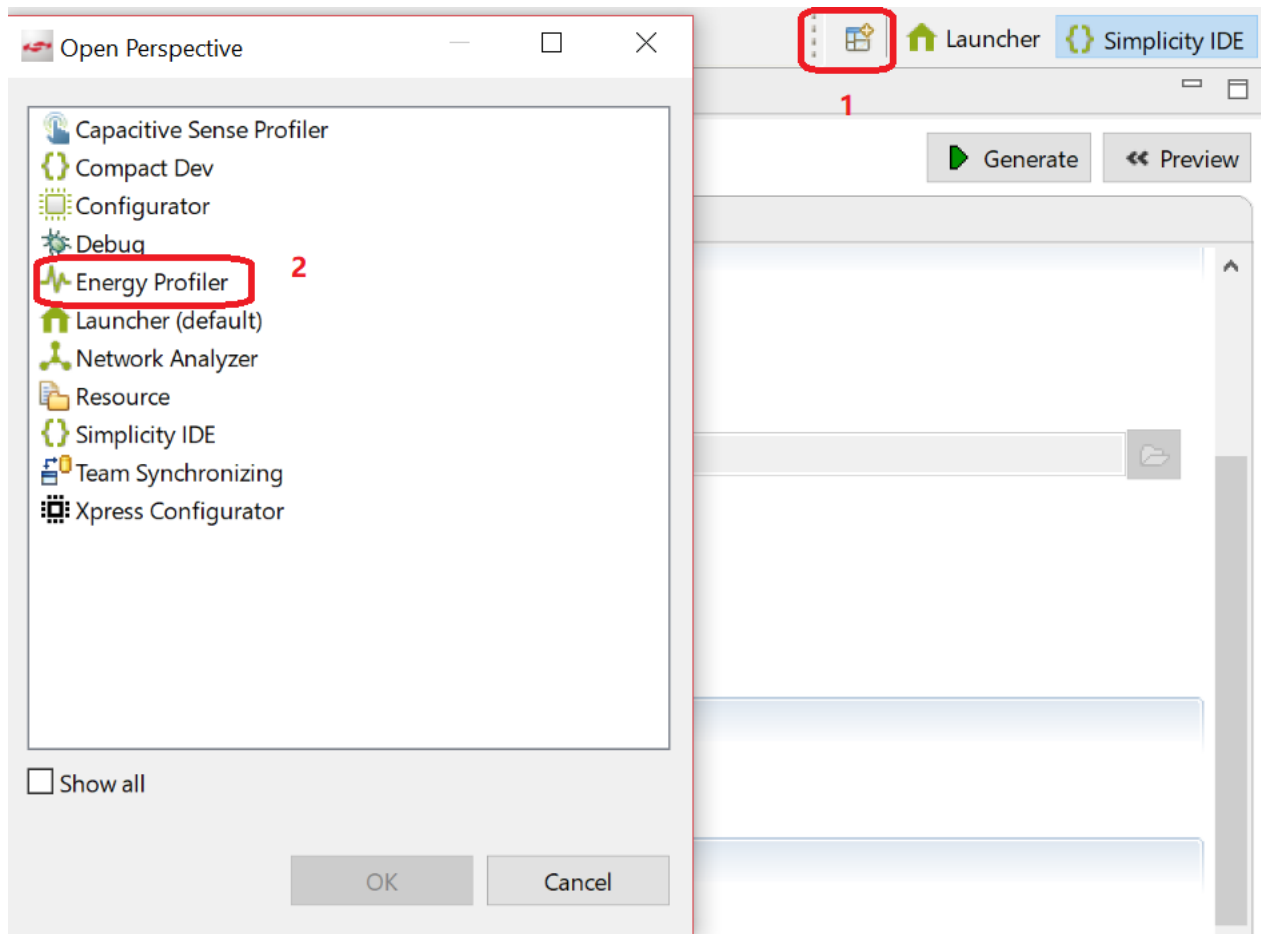


Figure 7 Start Energy Profiler

2. In the tool, on the left top, select "Quick Access", then select "Start Energy Profiler Capture";

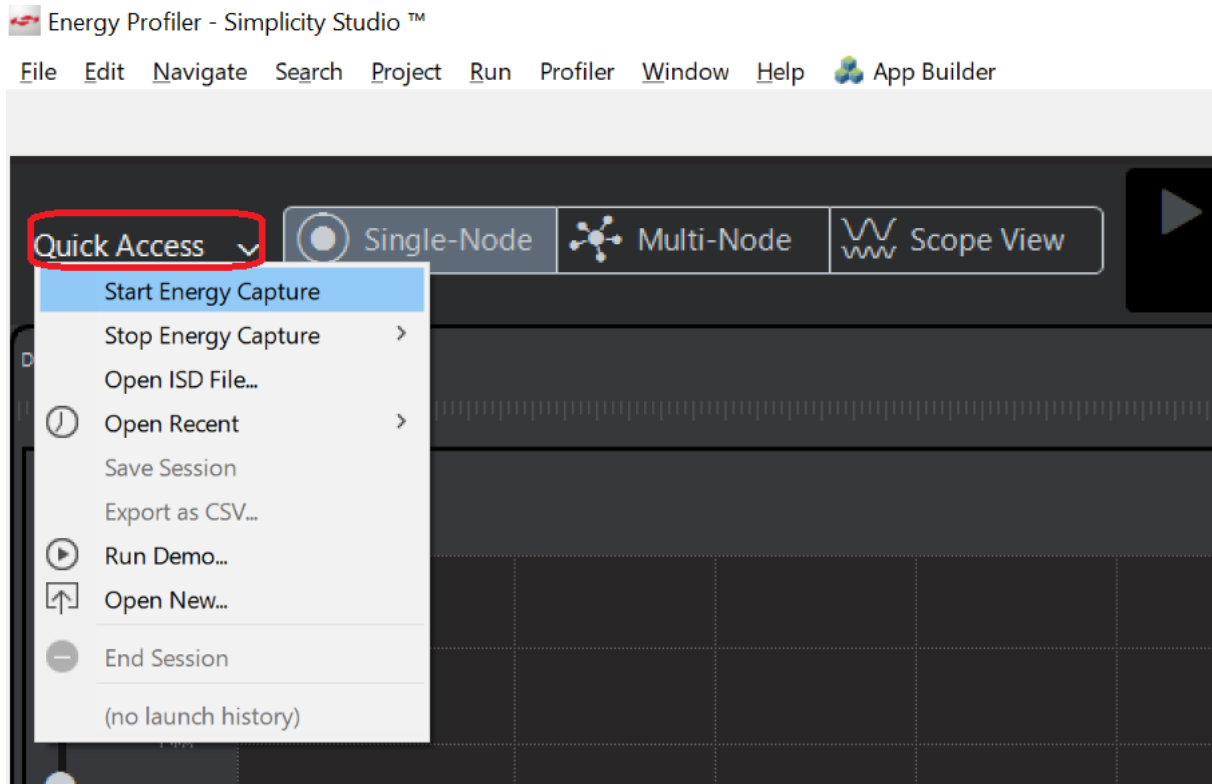


Figure 8 Start Energy Profiler Capture

3. Select your adapter;

6 Build the bootloader

1. Go to File -> New -> Project. This will bring up the New Project Wizard
2. Select "Silicon Labs AppBuilder Project". Click Next.
3. Select "Gecko Bootloader". Click Next.
4. Select the latest version. (Gecko Bootloader 1.9.2). Click Next.
5. Select "Internal Storage Bootloader(single image on 1MB devices)". Click Next.
6. Name your project(Whatever name you want). Click Next.
7. Select board and compiler. Then finish.

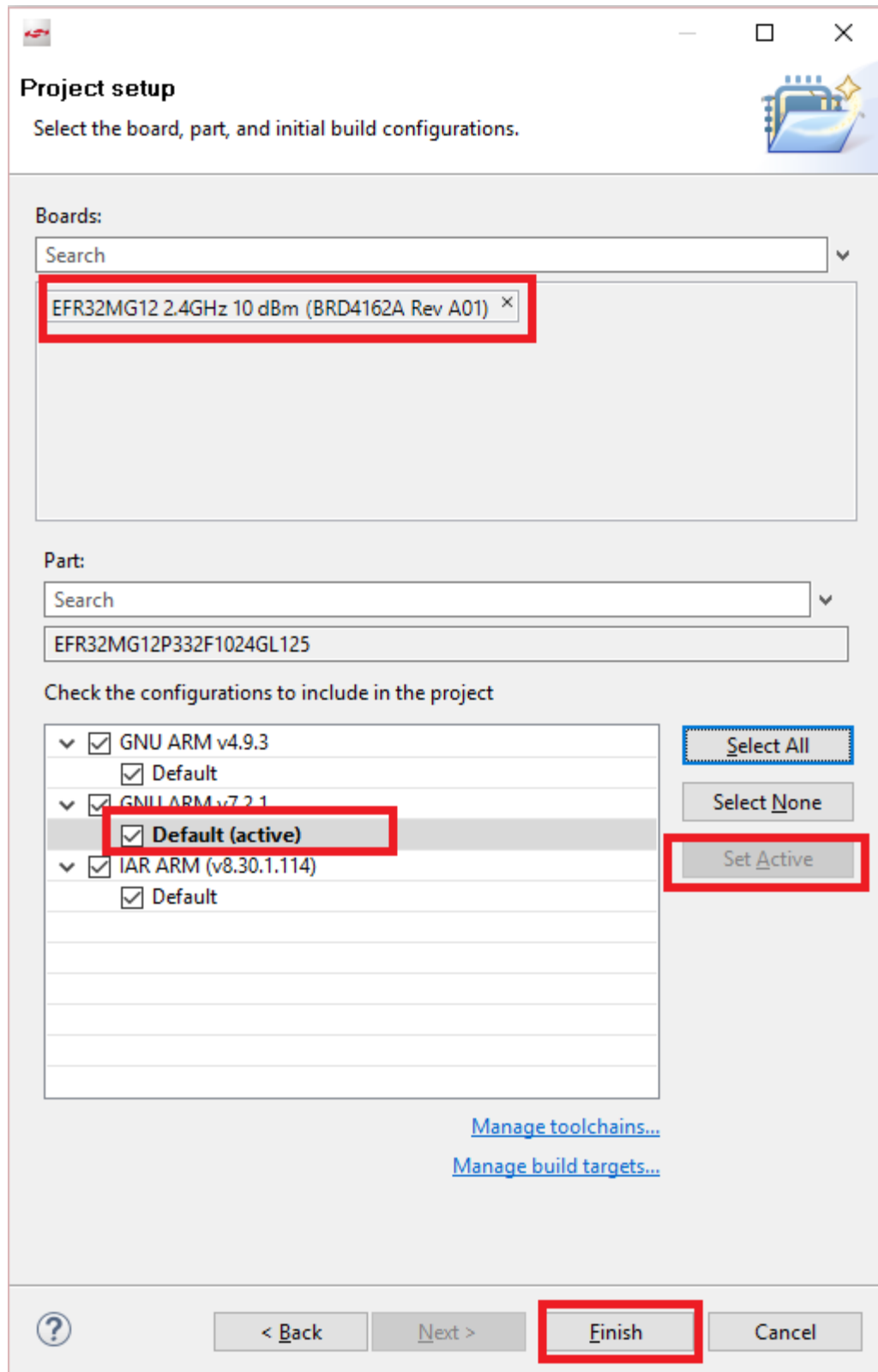



Figure 9 Select board and compiler

8. The new project should have been created now, with the project configuration file (an .isc file) open.
9. Click **"Generate"**. Notice the project files appearing in Project Explorer. A window saying Generation successful will appear. Click OK.

10. Select the project in Project Explorer window and compile your project by clicking on the Build icon . Ensure that the build completes with 0 errors.

7 Build the ZigBee sleepy end device

1. Go to File -> New -> Project. This will bring up the New Project Wizard
2. Select "Silicon Labs AppBuilder Project". Click Next.
3. Select "Silicon Labs Zigbee". Click Next.
4. Select our latest EmberZNet stack for SoC (in this case EmberZNet 6.6.4 GA SoC). Click Next.
5. On the bottom, select "ZigbeeMinimal". Click Next.
6. Name your project, such as "MyZSED". Click Next.
7. In next window (Project Setup), select board to BRD4162A, and compiler to "GCC v7.2" (If you don't have it, please install any other). Click Finish.

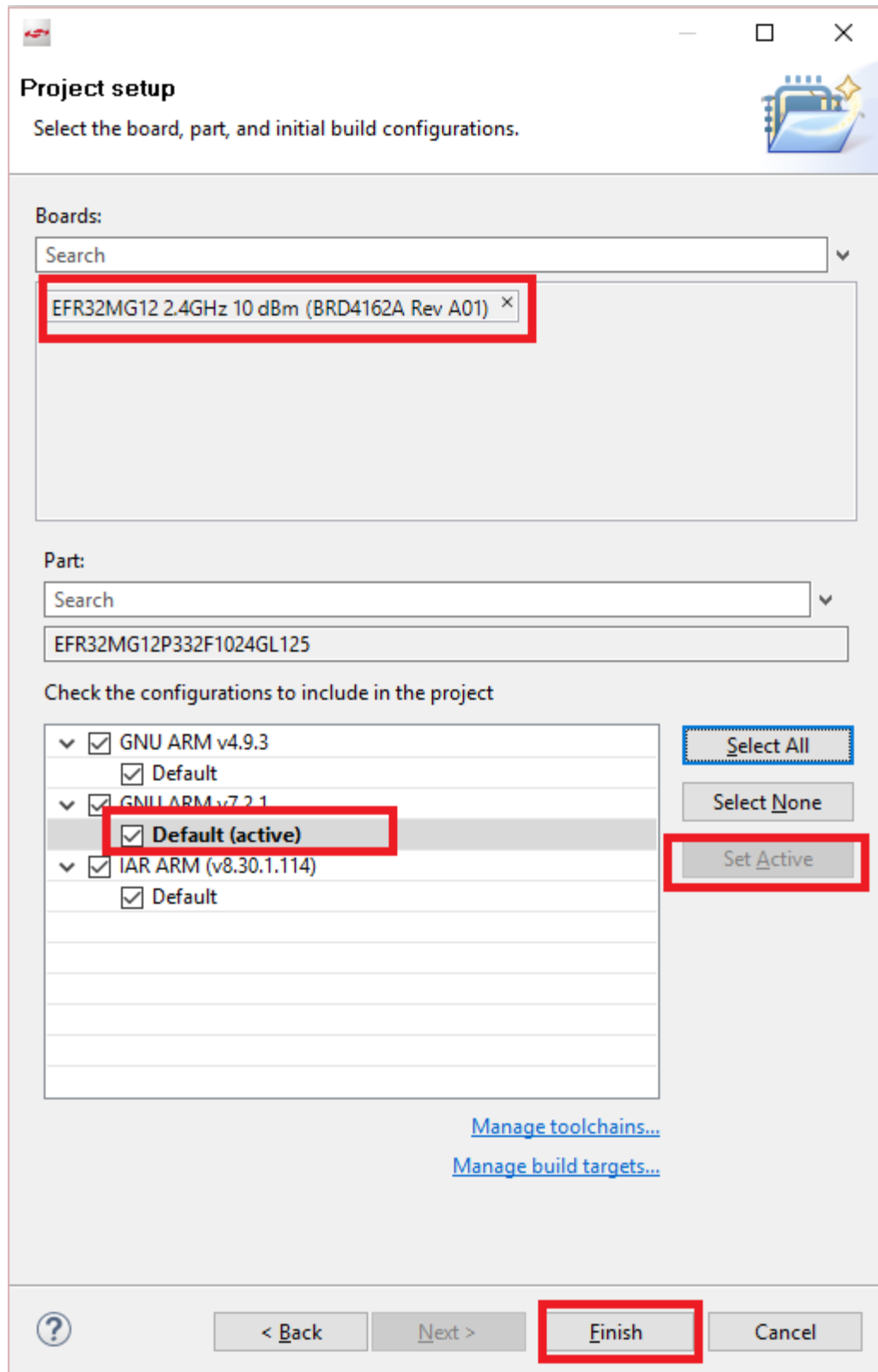


Figure 10 select board and compiler

8. The new project should have been created now, with the project configuration file (an .isc file) open.
9. Click "ZCL clusters" tab,
 - a. input 0x1049 in "Manufacture".
10. Click "Zigbee Stack" tab, select "Zigbee Device Type" to "Sleepy End Device".

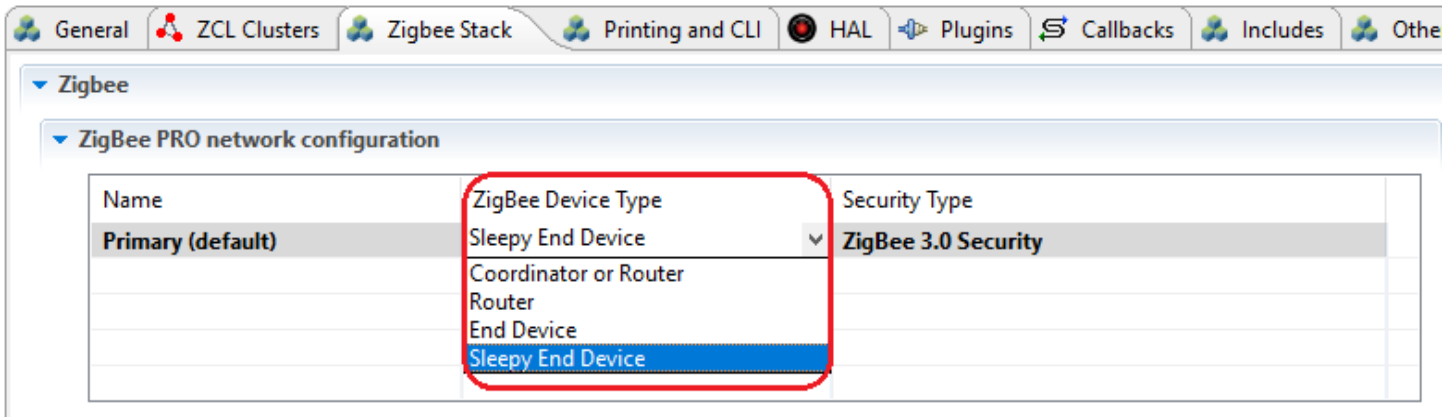


Figure 11 Select Zigbee device type

11. Click “Plugins” tab:

- a. Enable “Idle/Sleep” plugin, then in the properties of this plugin, enable the option “Stay awake when NOT joined”; So that when device hasn’t joined, the device can still keep awake, then you can use the command line to operate. Please also enable the option “Use button to force wakeup or allow sleep”. When the device is asleep, the command line interface won’t be available. With this interface, you can use button0 to force the device stay awake and use button1 to force the device allow sleep.

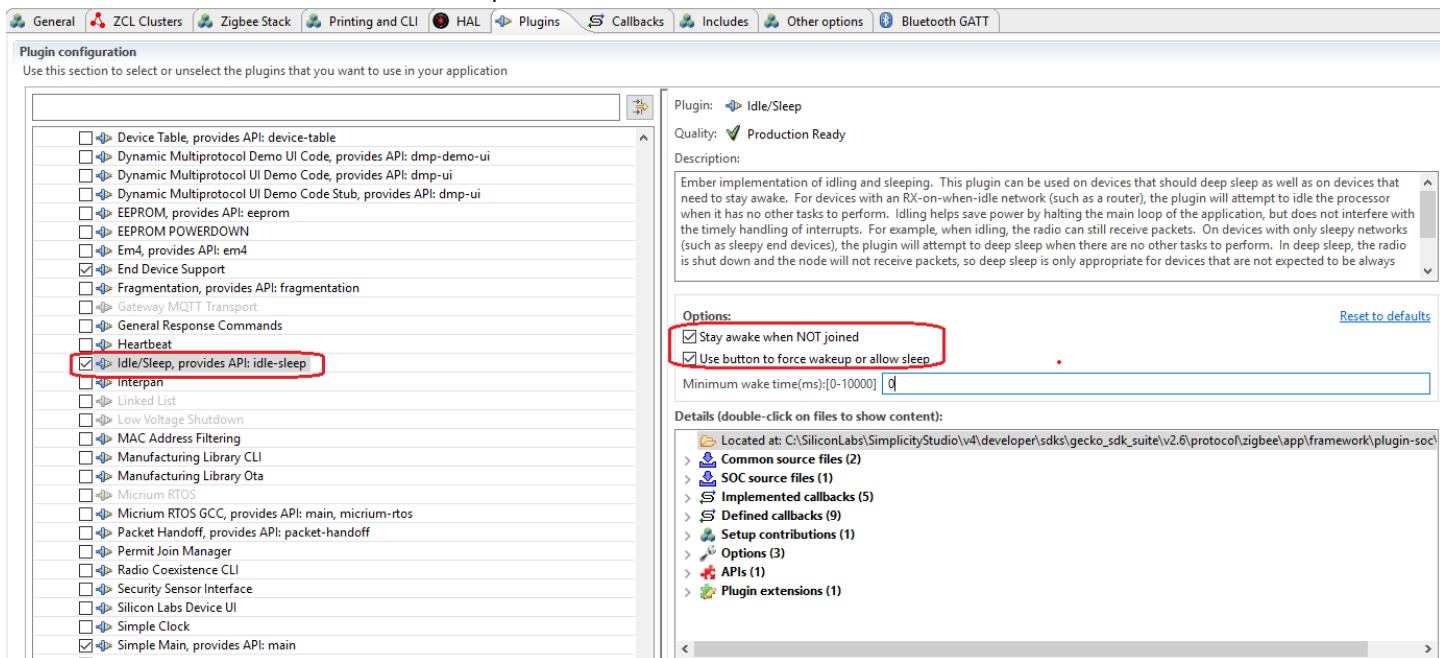


Figure 12 Plugin “Idle/Sleep”

- b. Select “End Device Support” plugin, in the properties, set the short poll interval to 1s and long poll interval to 10s.

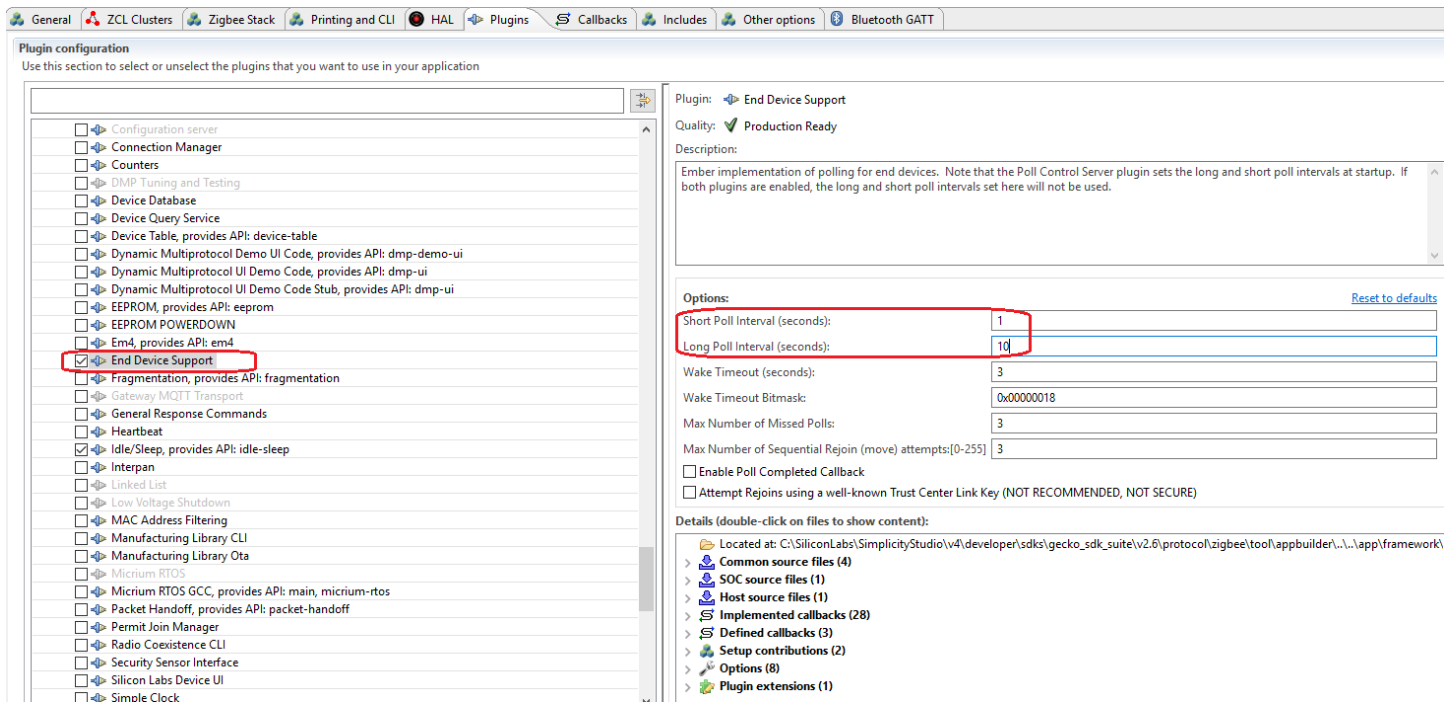



Figure 13 Plugin “End Device Support”

12. “Ctrl + S” to save the modified Project .ISC file, and click “Generate”. Notice the project files appearing in Project Explorer. A window saying “generating successfully” will appear. Click OK.
13. Select the project in Project Explorer window, and compile your project by clicking on the Build icon . Ensure that the build completes with 0 errors.

8 Test and observe the polling and the current.

1. Choose one WSTK as the sleepy end device, flash bootloader([please flash the xxx_combinded.s37](#)) to WSTK, and then flash sleepy end device application to WSTK
2. On the console of the switch, run following command to join the network

```
//Find a joinable network and join it
MyZSED> plugin network-steering start 0
```

3. Use info command to check whether you joined the right network

```
//Check the pan id and channel of ZSED. They should be the same as we provided.
MyZSED> info
MFG String:
AppBuilder MFG Code: 0x1002
node [(>)000B57FFFE648DD8] chan [18] pwr [3]
panID [0x2019] nodeID [0x0000] xpan [0x(>)A3E54612381CBF6B]
parentID [0xFFFF] parentRssi [0]
stack ver. [6.4.1 GA build 408]
nodeType [0x01]
Security level [05]
network state [02] Buffs: 73 / 75
Ep cnt: 2
ep 1 [endpoint enabled, device enabled] nwk [0] profile [0x0104] devId [0x0100] ver [0x00]
    in (server) cluster: 0x0000 (Basic)
    in (server) cluster: 0x0003 (Identify)
    in (server) cluster: 0x0004 (Groups)
    in (server) cluster: 0x0005 (Scenes)
    in (server) cluster: 0x0006 (On/off)
ep 2 [endpoint enabled, device enabled] nwk [0] profile [0x0104] devId [0x0100] ver [0x00]
    in (server) cluster: 0x0000 (Basic)
    in (server) cluster: 0x0003 (Identify)
    in (server) cluster: 0x0004 (Groups)
    in (server) cluster: 0x0005 (Scenes)
    in (server) cluster: 0x0006 (On/off)
Nwk cnt: 1
nwk 0 [Primary (pro)]
    nodeType [0x04]
    securityProfile [0x05]
```

4. If you joined a wrong network, please leave the network and join again.

```
//Leave network
MyZSED> network leave

//join again
MyZSED> plugin network-steering start 0
```

5. Make sure you have joined the right network before you continue.
6. Start a capture with Network Analyzer, observe the polling interval. Normally it should poll every 10 seconds.
7. Start Energy Profiler and measure the sleepy current. Record the average current.
8. Start a capture, then reset the sleepy end device. Observe the rejoin procedure in the sniffer.

After this, you have created a basic sleepy end device. Now we can continue to add our application.

9 Add application code

Now we are going to implement a temperature sensor which will read temperature from ADC and then report it to coordinator periodically. In this application, we will use the integrated ADC and its internal temperature sensor to read the temperature. To learn more about the ADC and temperature measurement, please refer to the reference manual.

1. Open the .isc file;
2. Click “ZCL clusters” tab,
 - a. In “ZCL device type” field, set “ZCL device type” to “HA Temperature Sensor”

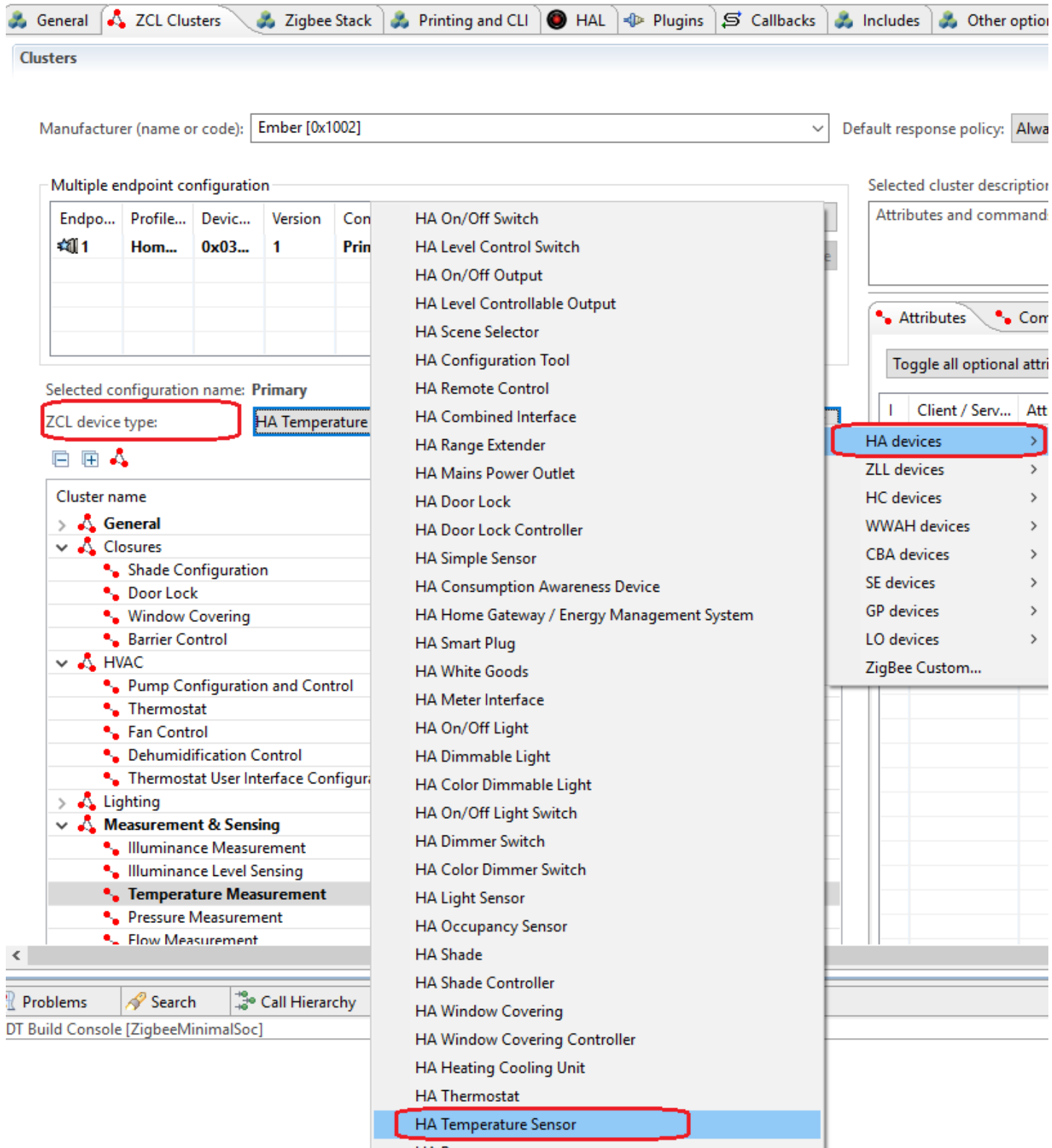


Figure 14 ZCL device type

- Make sure cluster "Temperature Measurement" server side is selected, then make sure the attribute "measured value" is selected. After that, turn to "Reporting" tab, and make sure the attribute "measured value" is selected. After this step, we can save temperature data in attribute "measured value" of cluster "Temperature Measurement", and we also can report this attribute to the coordinator.

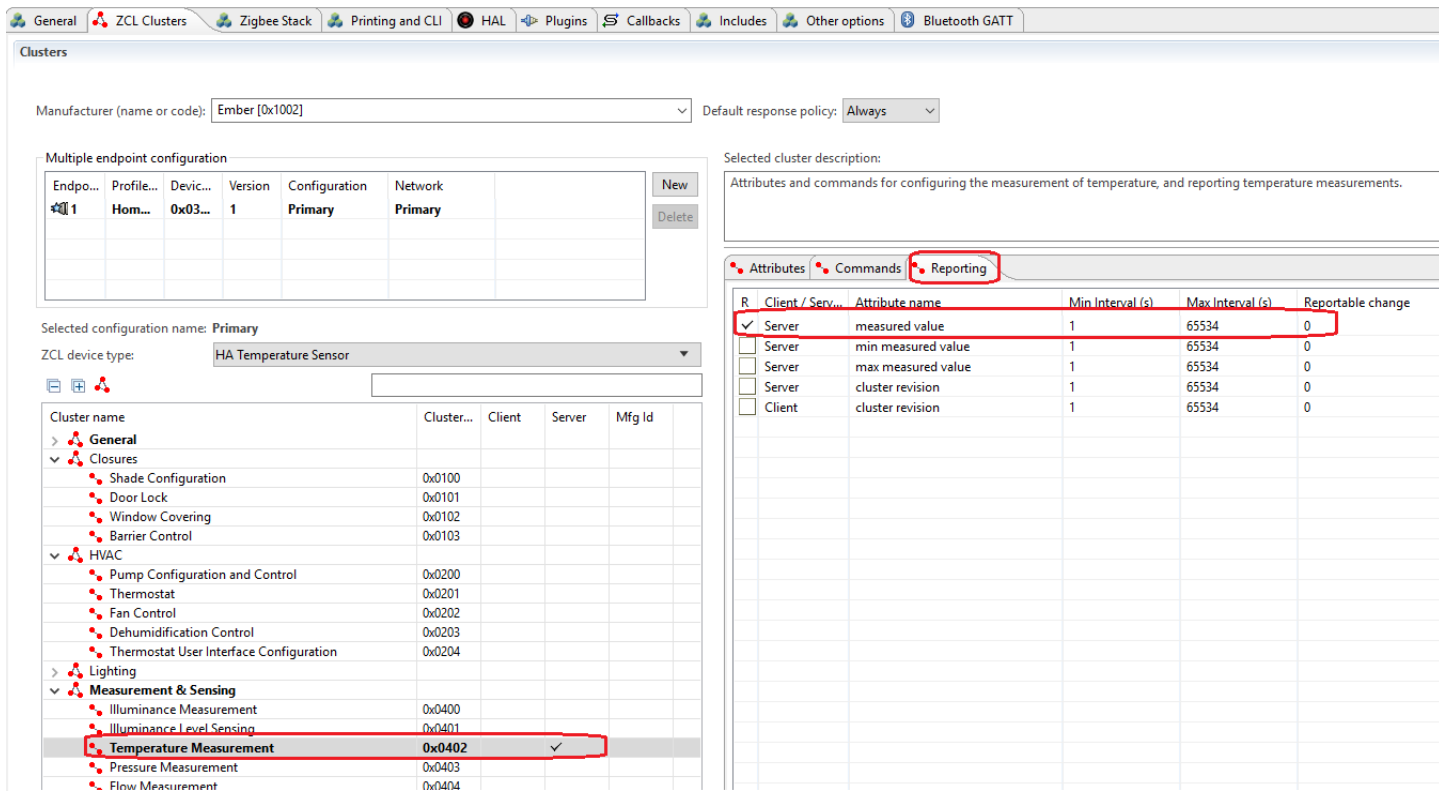


Figure 15 attribute and reporting

3. Click “Plugins” tab:

- Enable plugin “Reporting”, and in the properties, set reporting table size to 10. With this step, the attributes which will be reported will be saved in the reporting table, and will be reported periodically or reported after it is changed.

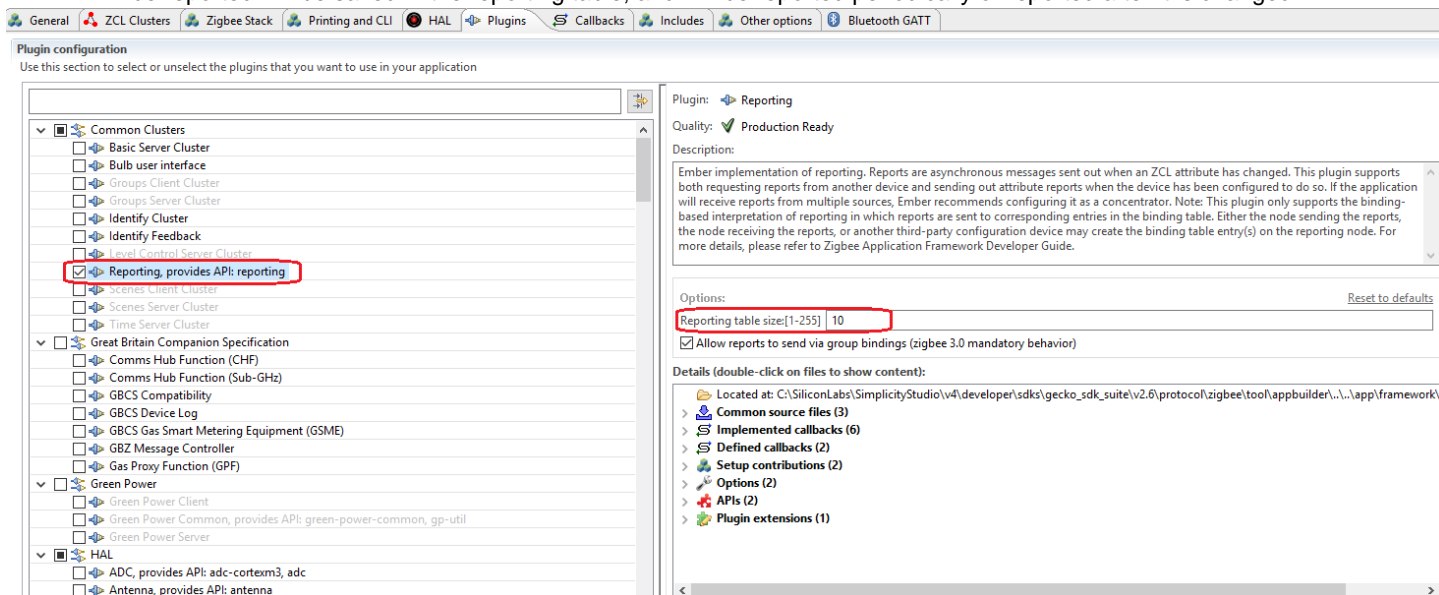


Figure 16 Plugin “Reporting”

- Enable plugin “Find and Bind Initiator”. With this step, the sleepy end device can start the “finding and binding” process, and with that, it can setup the binding table automatically. You can refer to Zigbee BDB spec section 8.5/8.6 to learn more about this procedure.

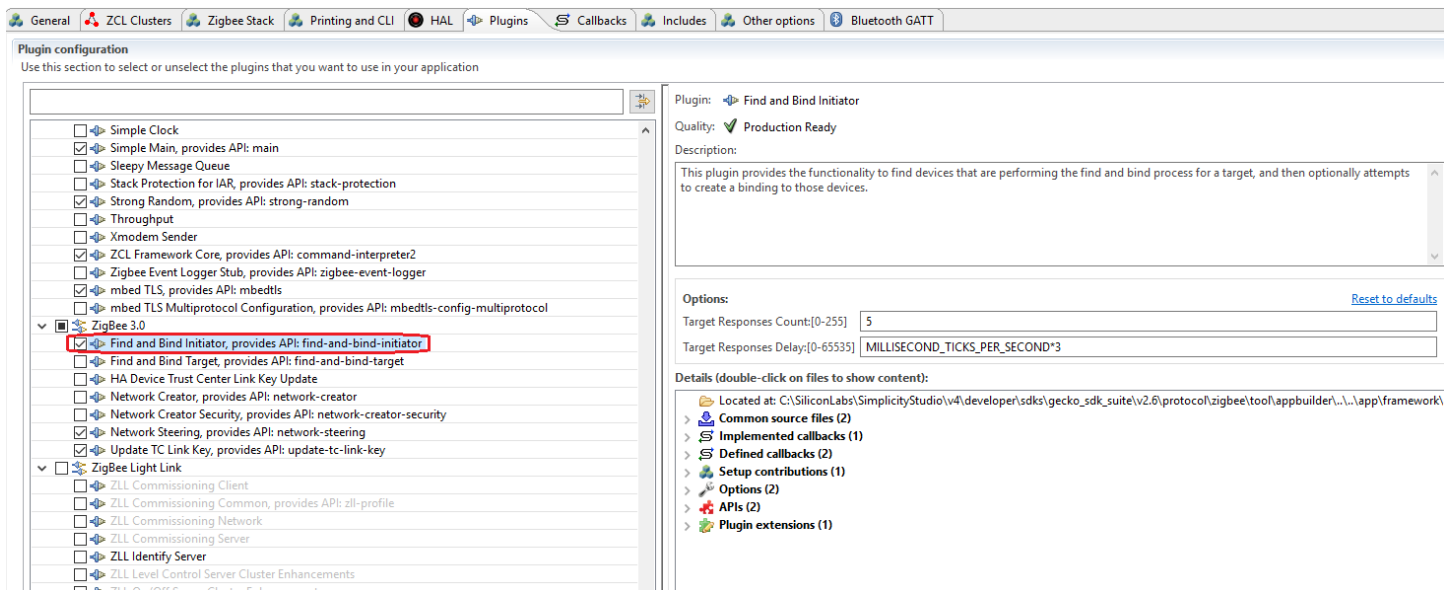


Figure 17 Plugin “Find and Bind Initiator”

14. Click “Callbacks” tab:

- a. Enable “Main Init” callback `emberAfMainInitCallback`; You can input “Main Init” in the filter to find it quickly.

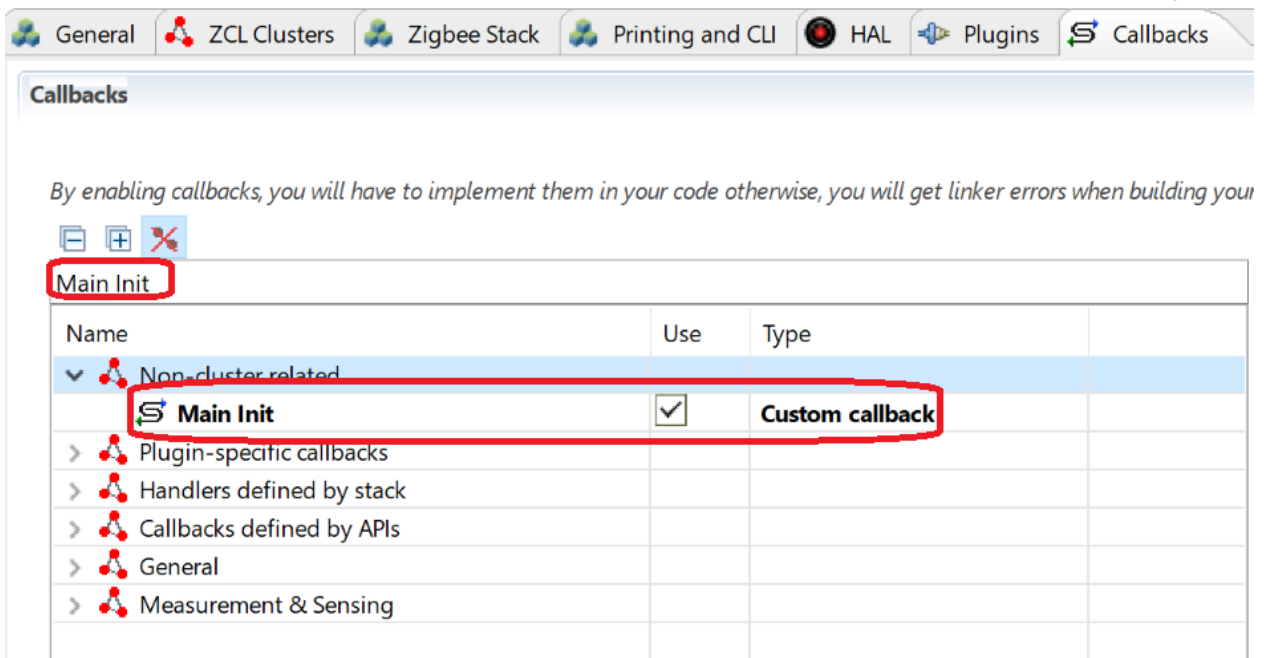


Figure 18 Callback “Main Init”

- b. Enable “Stack Status” callback `emberAfStackStatusCallback`;
 c. Enable “Complete” callback `emberAfPluginFindAndBindInitiatorCompleteCallback` of plugin find and bind initiator;

15. Click “Includes” tab, scroll to the bottom (You might need to scroll the bar on the very right as well), in the “Event Configuration” field, hit “New” button to add a custom event `customWriteAttributeEventData` and its handler `customWriteAttributeEventHandler`.

16. Save the modified Project .ISC file, and click “Generate”. Notice the project files appearing in Project Explorer. A window saying “generating successfully” will appear. Click OK.

17. Edit `<projectname>_callbacks.c`:

- a. Modify function `emberAfStackStatusCallback` as below:

```
bool emberAfStackStatusCallback(EmberStatus status)
{
    // This value is ignored by the framework.
}
```

```

    if (EMBER_NETWORK_UP == status) {
        //start find and bind procedure when joins network
        EmberStatus status = emberAfPluginFindAndBindInitiatorStart(1);
        emberAfCorePrintln("Find and bind initiator %p: 0x%X", "start", status);
    }

    return false;
}

```

b. Add function `emberAfPluginFindAndBindInitiatorCompleteCallback` as below:

```

void emberAfPluginFindAndBindInitiatorCompleteCallback(EmberStatus status)
{
    emberAfCorePrintln("Find and bind initiator %p: 0x%X", "complete", status);
}

```

c. Add the following source code snippets to read temperature from adc:

```

#include "em_adc.h"

EmberEventControl customWriteAttributeEventData;

static void AdcSetup(void)
{
    /* Enable ADC clock */
    CMU_ClockEnable(cmuClock_ADC0, true);

    /* Base the ADC configuration on the default setup. */
    ADC_Init_TypeDef init = ADC_INIT_DEFAULT;
    ADC_InitSingle_TypeDef sInit = ADC_INITSINGLE_DEFAULT;

    /* Initialize timebases */
    init.timebase = ADC_TimebaseCalc(0);
    init.prescale = ADC_PrescaleCalc(400000, 0);
    ADC_Init(ADC0, &init);

    /* Set input to temperature sensor. Reference must be 1.25V */
    sInit.reference = adcRef1V25;
    sInit.acqTime = adcAcqTime8; /* Minimum time for temperature sensor */
    sInit.posSel = adcPosSelTEMP;
    ADC_InitSingle(ADC0, &sInit);
}

static uint32_t AdcRead(void)
{
    ADC_Start(ADC0, adcStartSingle);
    while ( (ADC0->STATUS & ADC_STATUS_SINGLEDV) == 0 ) {
    }
    return ADC_DataSingleGet(ADC0);
}

static float ConvertToCelsius(int32_t adcSample)
{
    uint32_t calTemp0;
    uint32_t calValue0;
    int32_t readDiff;
    float temp;

    /* Factory calibration temperature from device information page. */
    calTemp0 = ((DEVINFO->CAL & _DEVINFO_CAL_TEMP_MASK)
        >> _DEVINFO_CAL_TEMP_SHIFT);

    calValue0 = ((DEVINFO->ADC0CAL3
        /* _DEVINFO_ADC0CAL3_TEMPREAD1V25_MASK is not correct in
        current CMSIS. This is a 12-bit value, not 16-bit. */

```

```

        & 0xFFFF0)
        >> _DEVINFO_ADC0CAL3_TEMPREAD1V25_SHIFT);

if ((calTemp0 == 0xFF) || (calValue0 == 0xFFFF)) {
    /* The temperature sensor is not calibrated */
    return -100.0;
}

/* Vref = 1250mV
   TGRAD_ADCTH = 1.84 mV/degC (from datasheet)
   */
readDiff = calValue0 - adcSample;
temp      = ((float)readDiff * 1250);
temp      /= (4096 * -1.84);

/* Calculate offset from calibration temperature */
temp      = (float)calTemp0 - temp;
return temp;
}

void customWriteAttributeEventHandler()
{
    int32_t sample = 0;
    int16_t temp = 0;

    emberEventControlSetInactive(customWriteAttributeEventData);


    sample = AdcRead();
    temp = ConvertToCelsius(sample);
    emberAfCorePrintln("sample=%d", sample);
    emberAfCorePrintln("temp=%d", temp);

    emberAfWriteServerAttribute(1,
                                ZCL_TEMP_MEASUREMENT_CLUSTER_ID,
                                ZCL_TEMP_MEASURED_VALUE_ATTRIBUTE_ID,
                                &temp,
                                ZCL_INT16S_ATTRIBUTE_TYPE);

    emberEventControlSetDelayMS(customWriteAttributeEventData, 30000);
}

void emberAfMainInitCallback(void)
{
    AdcSetup();
    emberEventControlSetDelayMS(customWriteAttributeEventData, 30000);
}

```

18. Select the project in Project Explorer window, and compile your project by clicking on the Build icon . Ensure that the build completes with 0 errors.

10 Test and observe the capture and current

1. Flash the new sleepy end device application to WSTK;
2. Connect the WSTK, right click and then select "Make a capture";

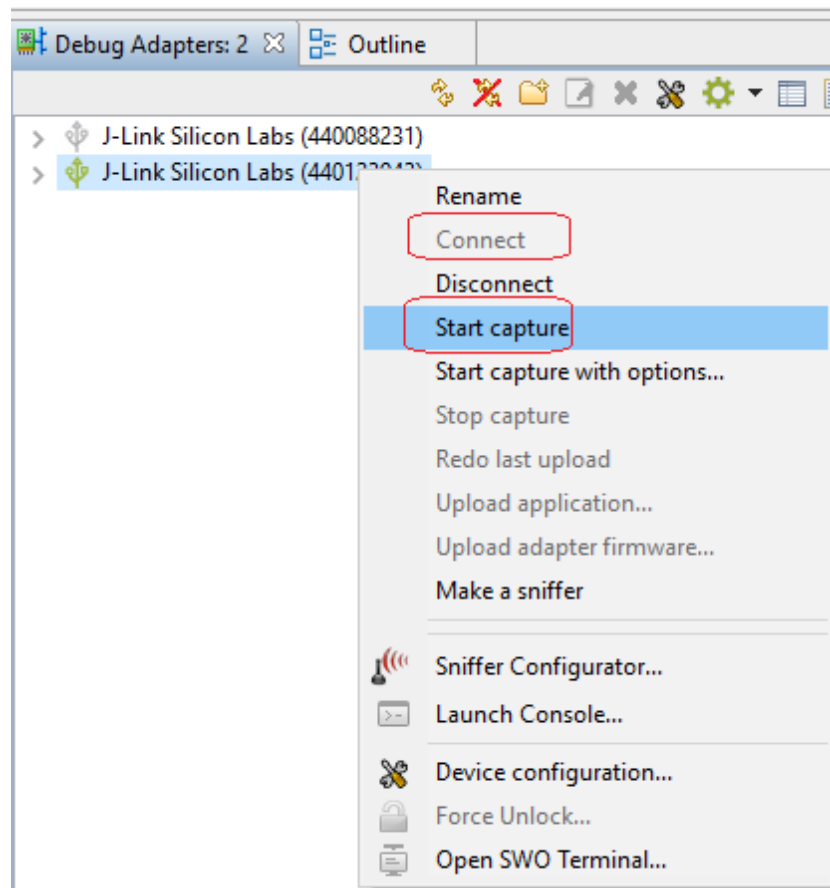


Figure 19 Make a capture

3. Observe the periodically report and also the polling at this stage;

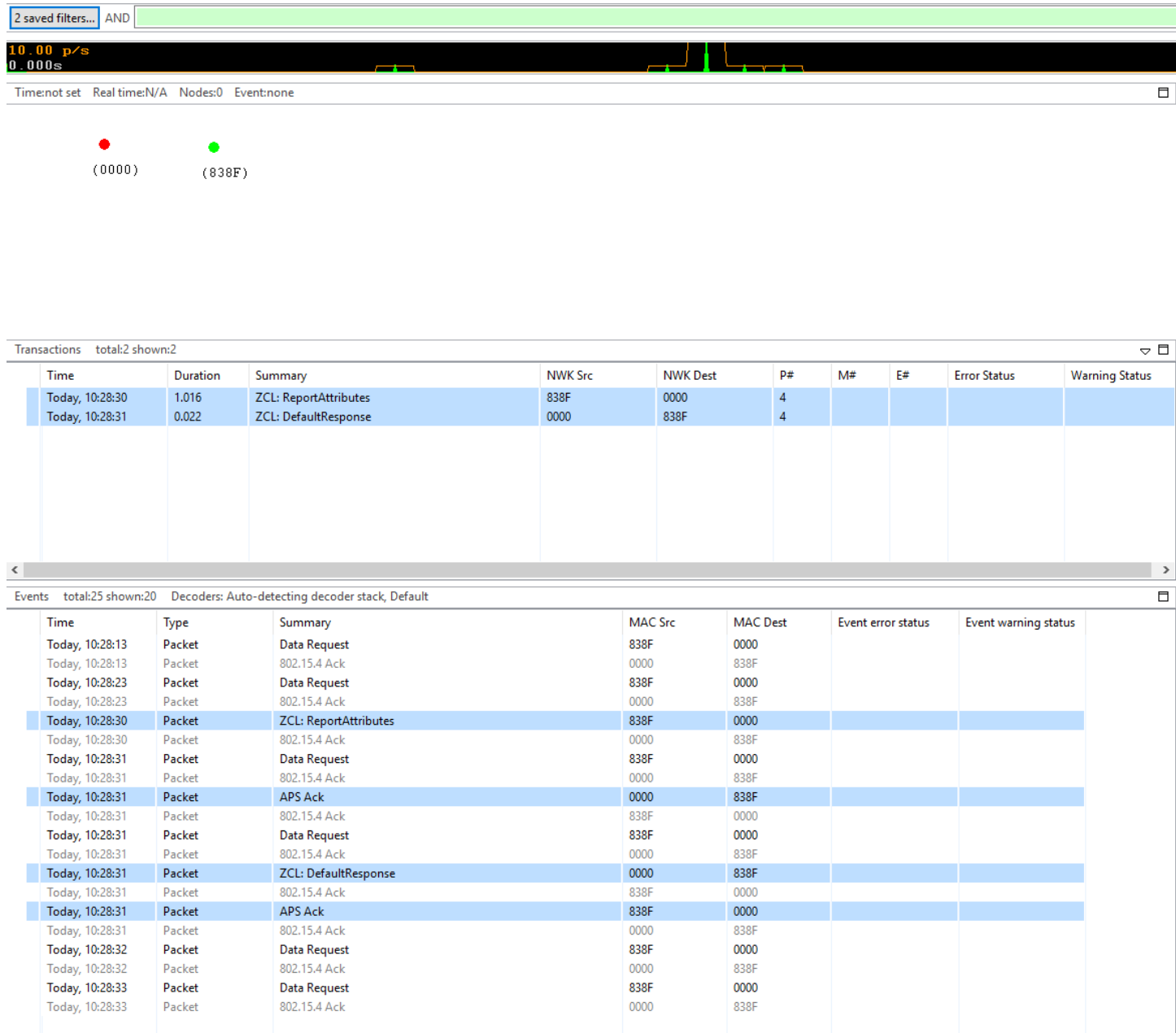


Figure 20 Observe the report and polling

4. Observe the current at this stage;

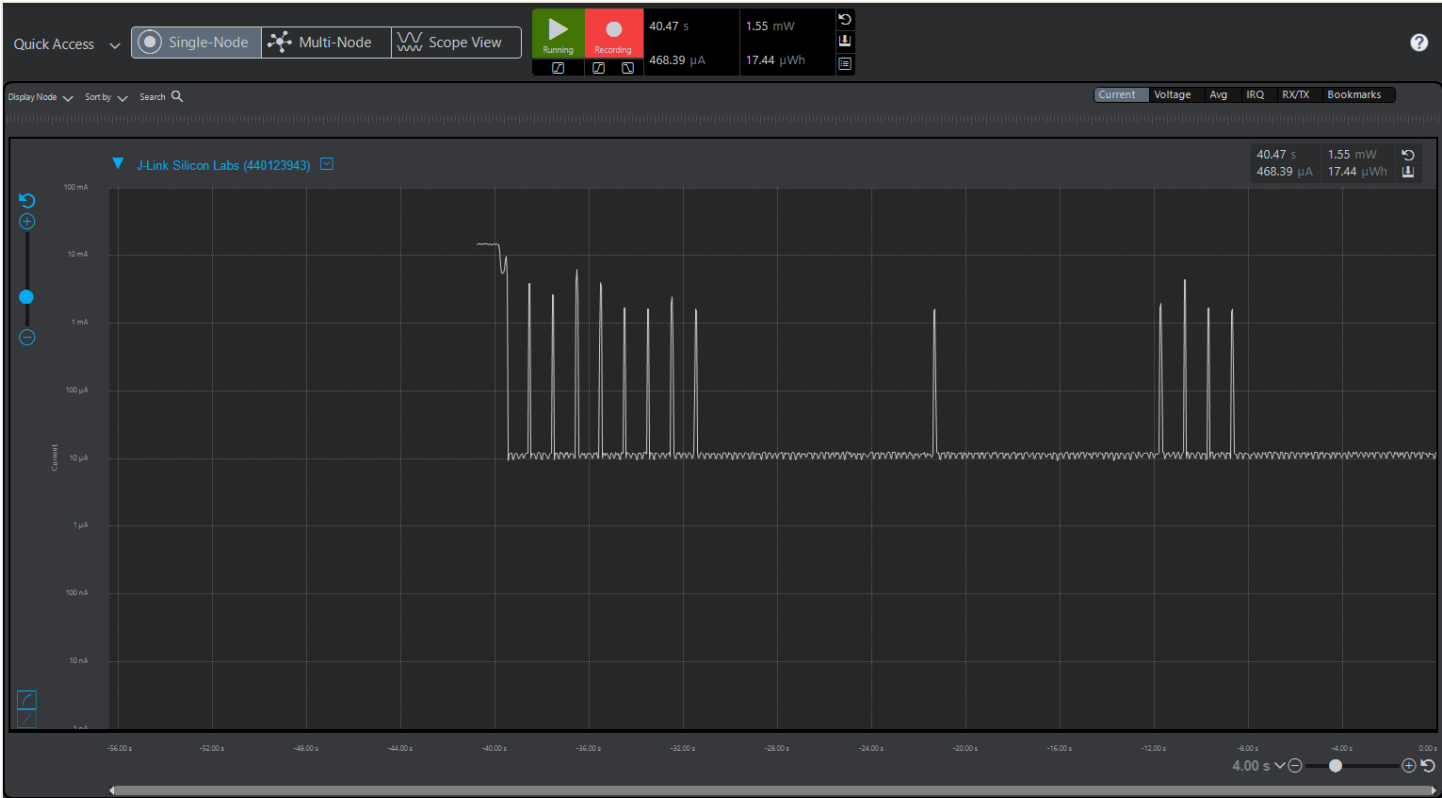


Figure 21 Observe the current