实验手册：基于芯科科技 GSDK 开发 Matter 灯和开关等设备

Silicon Labs EFR32MG24 Breakout Board 评估板是一使用方便和低成本套件，可以用于 Matter Thread SoC 应用评估以及固件开发。

EFR32MG24 评估板上引出了 ***Mini-Simplicity*** 调试编程接口。Simplicity Studio IDE 可通过此接口进行固件烧写和在线代码级调试；同时也可以通过引脚 ***SWCLK\SWDIO\3V3\GND*** 接口进行烧录。

实验目的

- 熟悉芯科科技全线产品开发环境 Simplicity Studio
- 了解 Matter，对 Matter 协议不再陌生
- 掌握如何基于芯科科技 IoT 平台开发 Matter 设备

EFR32MG24 评估板结合基于 EFR32MG21 的 USB Stick、J-Link 工具以及 Simplicity Studio 可以构成一套低成本的 Matter SoC 固件应用调试与开发工具。

Table of Contents

1 预备知识

本实验中涉及的专有名词及相关知识：

- 对 Ubuntu 开发环境有一定的使用基础
- 对 OpenThread 协议有一定的了解
- 对 C++有一定的基础
- 对 Matter 协议规范有个初步认识
- 名词解释
  - OT-RCP：Open Thread Radio Co-Processor。Thread 无线协处理器
  - OTBR：Open Thread Board Router。Thread 边界路由器
  - chip-tool：Linux 应用程序。用于 Matter 协议控制
  - ot-ctl：Thread 网络控制的应用程序

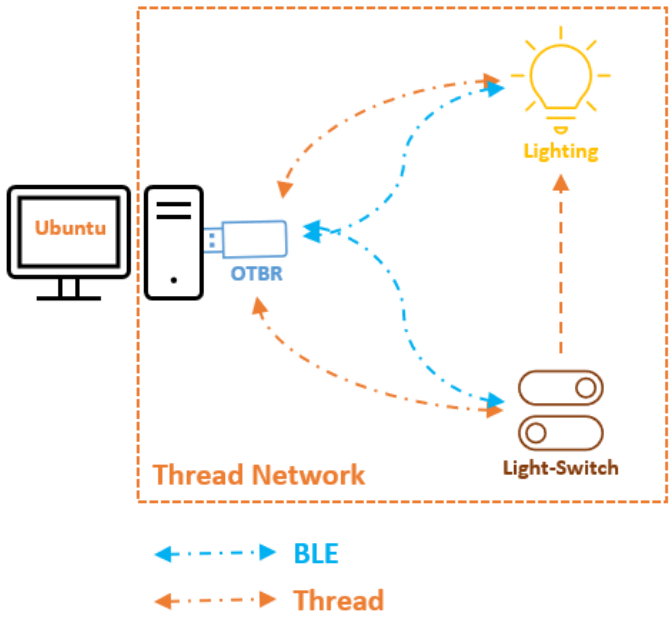
2 实验目的

通过本次实验，开发者可以理解并掌握以下内容：

- 熟悉芯科科技全线产品的开发环境 Simplicity Studio v5
- 了解 Matter，对 Matter 协议不再陌生
- 让开发者认为开发一全 Matter 设备的门槛也不是那么高
- 掌握如何基于芯科科技 IoT 平台开发 Matter 设备

3 实验内容

搭建一个基本的 Matter over Thread 网络，如下图所示：



通过本次《使用 GSDK 开发 Matter 开关和 Matter 灯》实验，开发者可以理解并掌握以下内容：

- 基于芯科科技 GSDK 开发一个 Matter 灯
- 基于芯科科技 GSDK 开发一个 Matter 开关
- 基于芯科科技 GSDK 开发一个 bootloader
- 基于芯科科技 GSDK 开发一个 OT-RCP
- 通过终端命令行控制 Matter 灯
- 使用 Matter 开关控制 Matter 灯

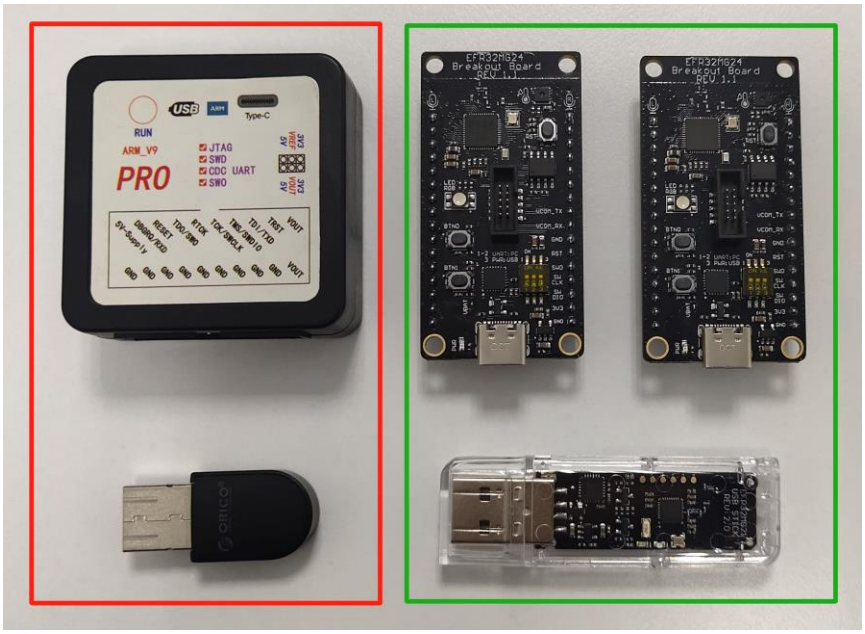
4 实验准备

4.1 硬件准备

- 安装 Ubuntu 虚拟机的电脑一台
- MG24 Breakout 开发板两块
- USB Stick (用作 OTBR)
- USB 线一条
- J-Link 烧录器一台
- BLE Dongle 一块
- 树莓派一台（可选）

本次培训硬件清单：

- 红框部分：需要开发者自行购买
  - JLINK (为 Breakout Board 烧录程序和 log 输出)
  - BLE Dongle (配置 Matter 设备入网时会用到 BLE 功能)
    - 点击查看测试过的 BLE Dongle 型号
- 绿框部分：芯科科技向通过本次培训审核的工程师免费提供
  - Breakout Board x2（用于开发 Matter 设备）
  - USB Stick（OT-RCP 模式，用于开发 OTBR）



Breakout 开发板用户接口说明（工程默认配置）：

用户接口	芯片接口	Lighting 演示	Light-Switch 演示
BTN0	PB02	功能键	功能键
BTN1	PB03	灯开关控制键	开关键
RGB	Low Level PD02(R) PA04(G) PB00(B)	状态指示灯：红色 受控 LED 灯：绿色	状态指示灯：红色

4.2 软件准备

芯科科技将提供一个 ubuntu 镜像，已完全搭建好 Matter 开发环境，开发者可直接使用。

若需自行搭建环境的，可参考Matter 开发环境和 Matter SDK 环境两个章节。

4.2.1 Matter 开发环境

Matter 设备开发需要用到 Windows 环境和 Linux 环境，可根据实际情况进行选择。

- Windows 环境
  - Simplicity Commander
    - 烧录设备固件
    - 转换固件格式（比如：将 s37 文件转换成 gbl 文件）
  - Simplicity Studio
    - 创建设备工程、编译固件
- Linux 环境：Ubuntu
  - OTBR 控制环境
  - 如果是桌面环境，也可以安装 Simplicity Studio 和 Simplicity Commander
- Linux 环境：Raspberry Pi 4B (可选)
  - OTBR 控制环境
  - 芯科科技提供的系统镜像：SilabsMatterPi
    - 集成了 Matter SDK 和开发环境
    - 默认用户名和密码都为：ubuntu
- Linux 环境依赖安装

```
$ sudo apt-get install git gcc g++ pkg-config libssl-dev libdbus-1-dev libglib2.0-dev libavahi-client-dev
ninja-build python3-venv python3-dev python3-pip unzip libgirepository1.0-dev libcairo2-dev libreadline-dev
# 安装实用工具
$ sudo apt-get install net-tools openssh-server openssh-client
# 启用双向复制粘贴功能
$ sudo apt-get install virtualbox-guest-x11
$ sudo VBoxClient --clipboard
```

4.2.2 Matter SDK 环境

从 silicon labs 官方 github 获取 Matter 代码：

```
$ git clone https://github.com/SiliconLabs/matter.git
$ cd matter
# 同步子模块（受网速影响，可能需要比较长的时间）
$ ./scripts/checkout_submodules.py --shallow --recursive --platform efr32
# 检测并完善编译环境（对网络有要求，需要访问国外网站。受网速影响，可能需要约 1 小时左右）
$ source scripts/activate.sh
```

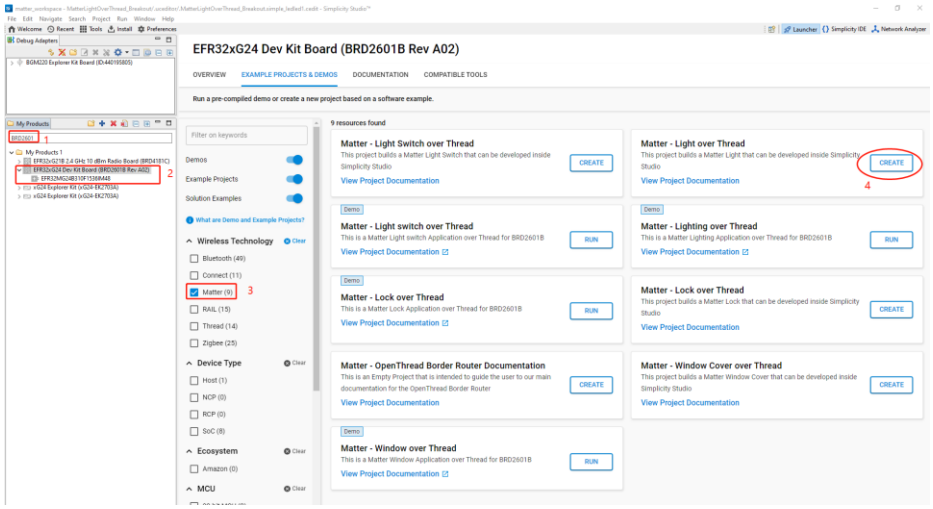
注：这个过程需要同步较多子模块，并且有些需要访问到国外网站，整个过程可能耗时可能在数小时或更长时间。这里推荐使用芯科科技提供的 ubuntu 镜像文件。

5 实验步骤

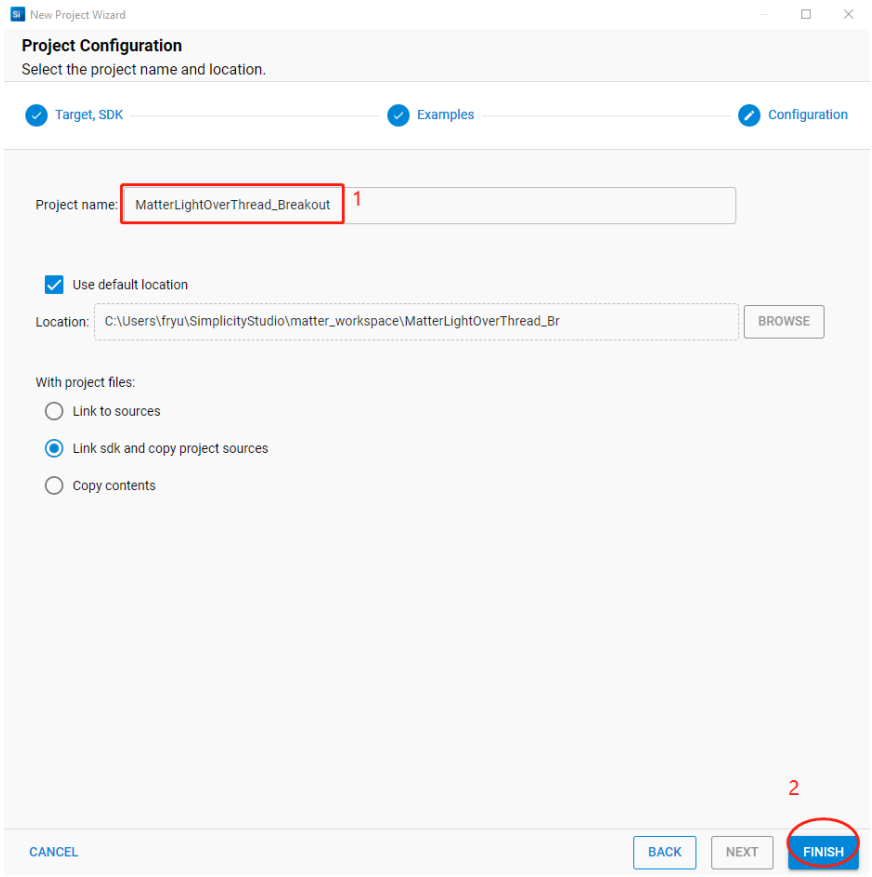
5.1 基于 GSDK 开发 Matter 灯

5.1.1 创建&编译工程

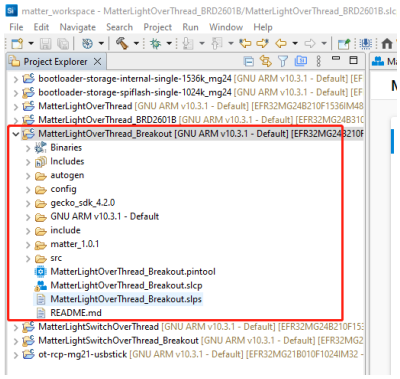
1. 选择`BRD2601`作为基础，创建`Matter-Light over Thread`工程。



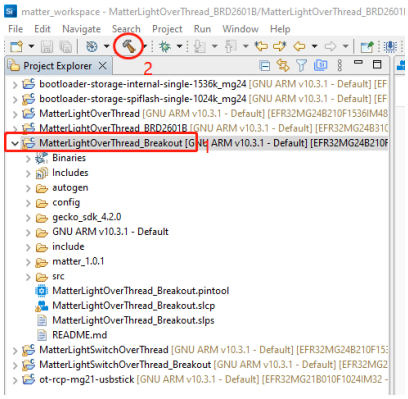
2. 修改工程名为`MatterLightOverThread_Breakout`并创建。（改工程名不是必须的）



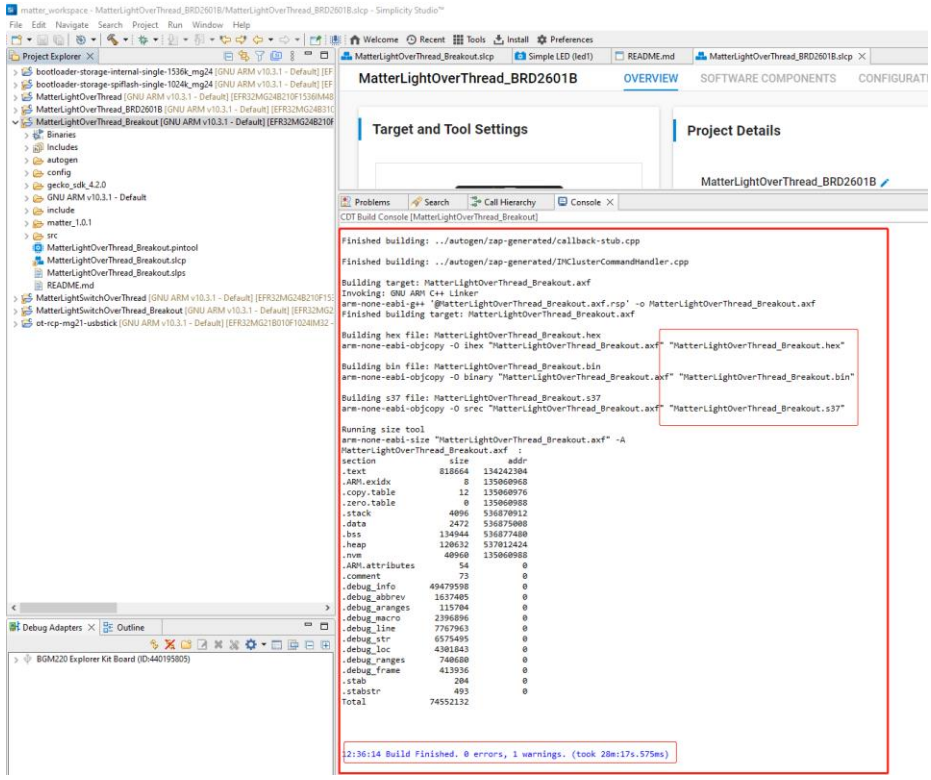
3. 工程创建完成后可看到如下图所示



4. 编译工程。选择工程，点击`锤子图标`进行编译。



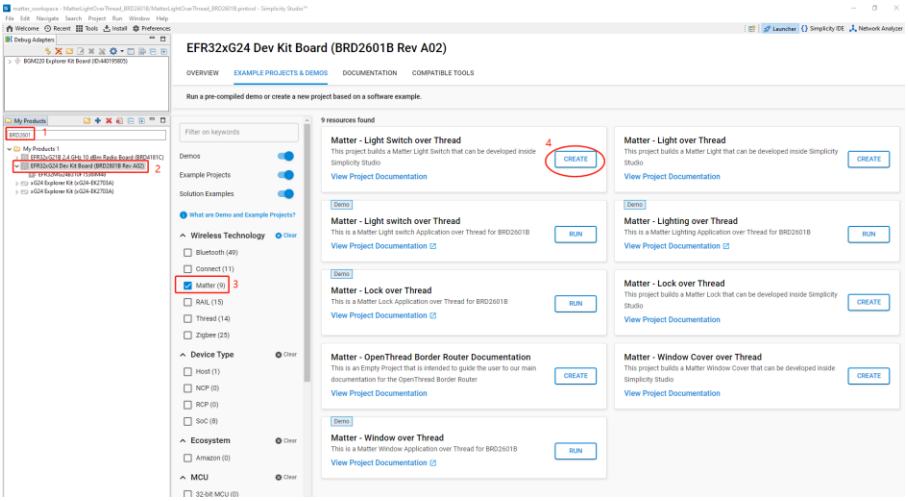
5. 编译完成后如下图所示。可看到编译出的固件（.s37/.bin/.hex）和编译时间。



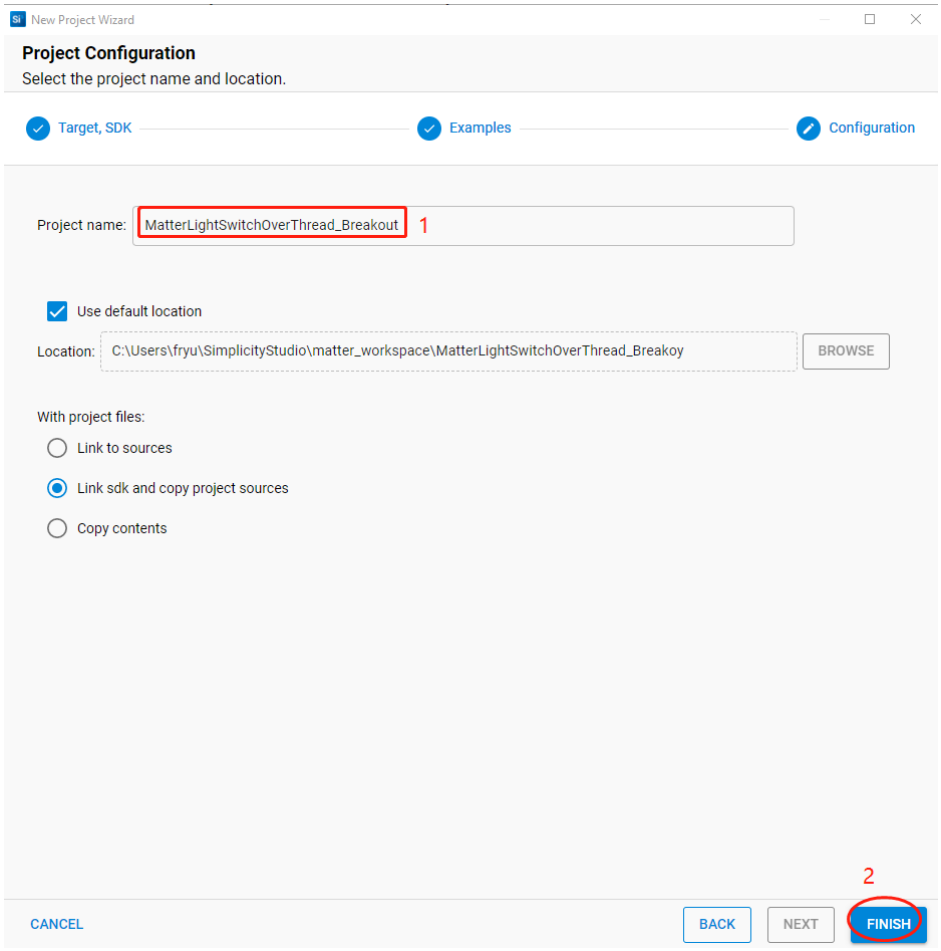
5.2 基于 GSDK 开发 Matter 开关

5.2.1 创建&编译工程

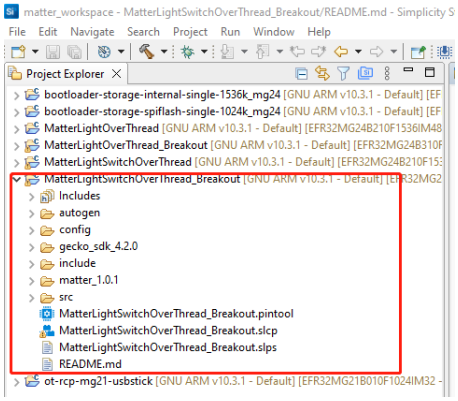
1. 选择`BRD2601`作为基础，创建`Matter-Light Switch over Thread`工程。



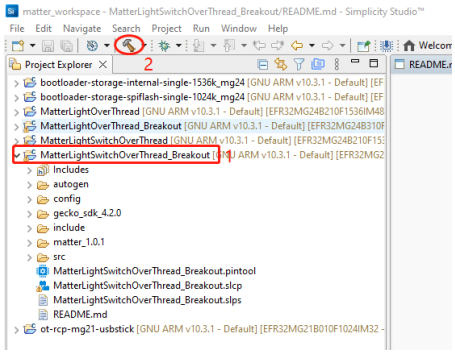
2. 修改工程名为`MatterLightswitchOverThread_Breakout`并创建。（改工程名不是必须的）



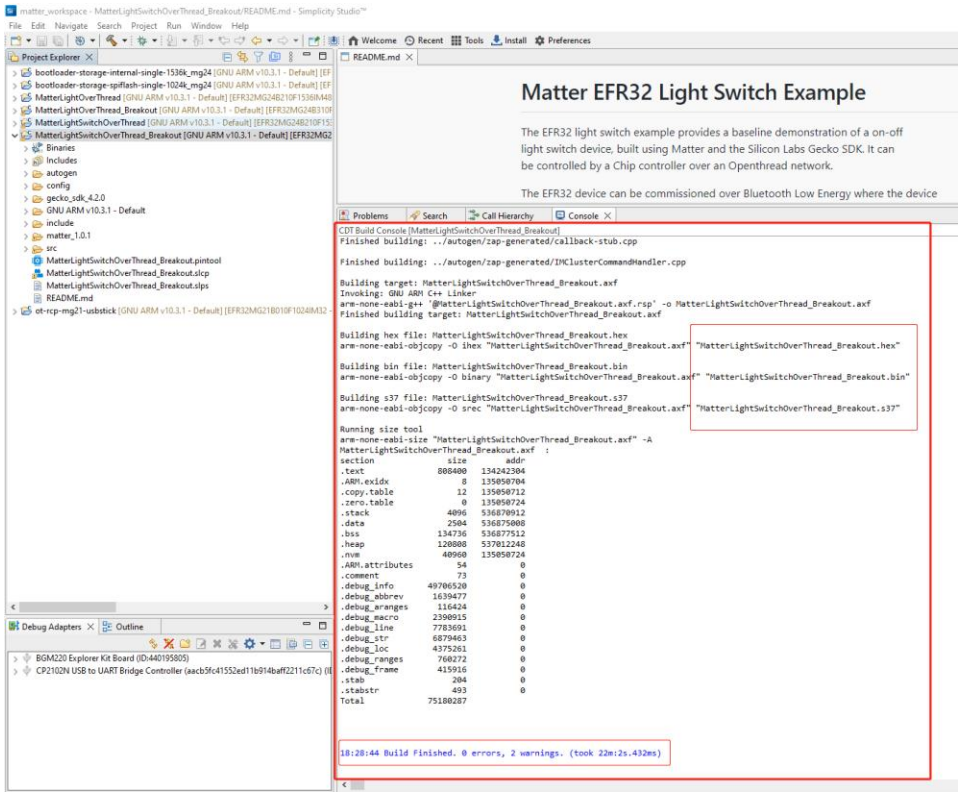
3. 工程创建完成后可看到如下图所示



4. 编译工程。选择工程，点击`锤子图标`进行编译。



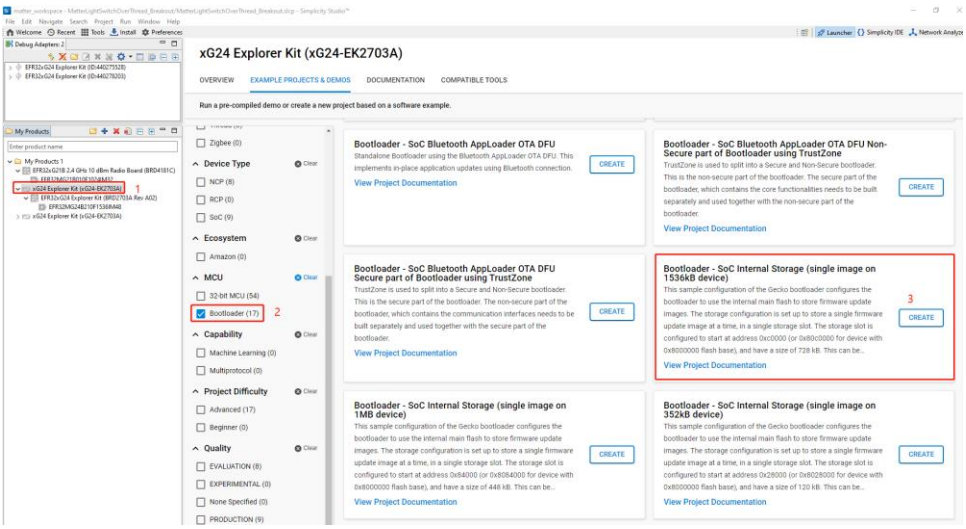
5. 编译完成后如下图所示。可看到编译出的固件（.s37/.bin/.hex）和编译时间。



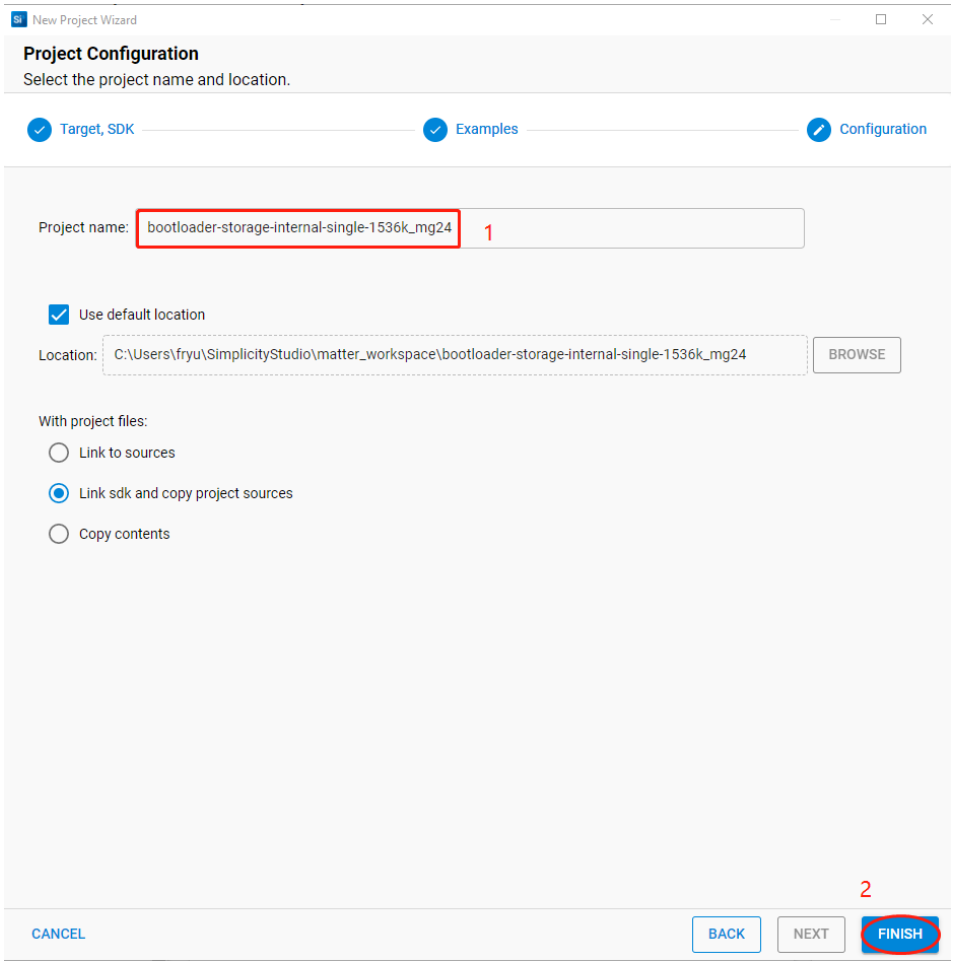
5.3 基于 GSDK 开发 bootloader

5.3.1 创建&编译工程

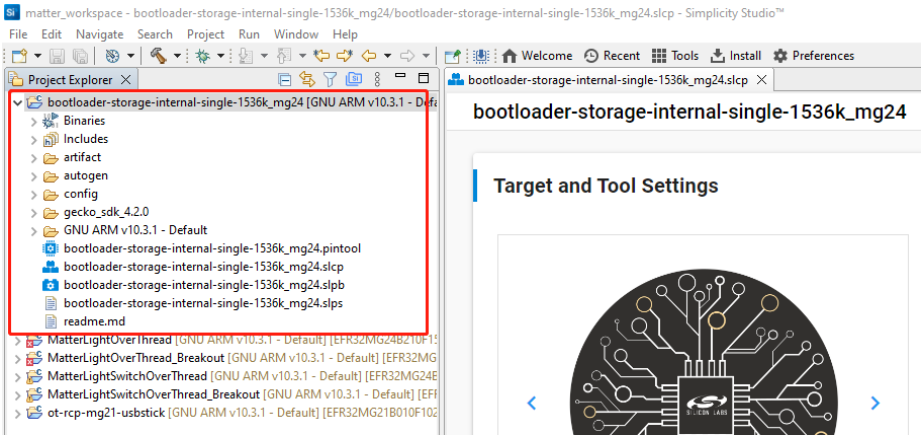
1. 选择`Bootloader-SoC Internal Storage(single image on 1536kB device)`示例创建工程。



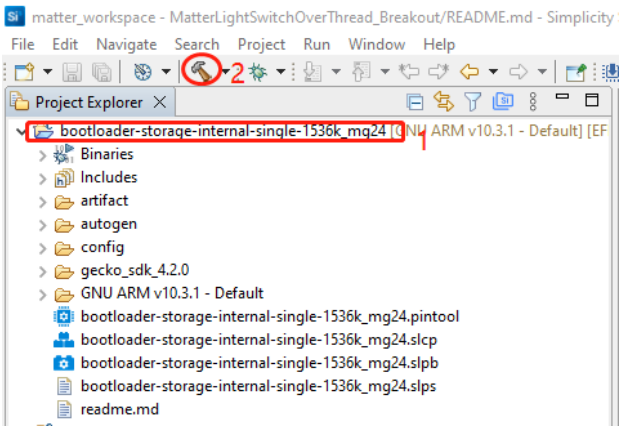
2. 修改工程名为`bootloader-storage-internal-singal-1536k_mg24`并创建。（改工程名不是必须的）



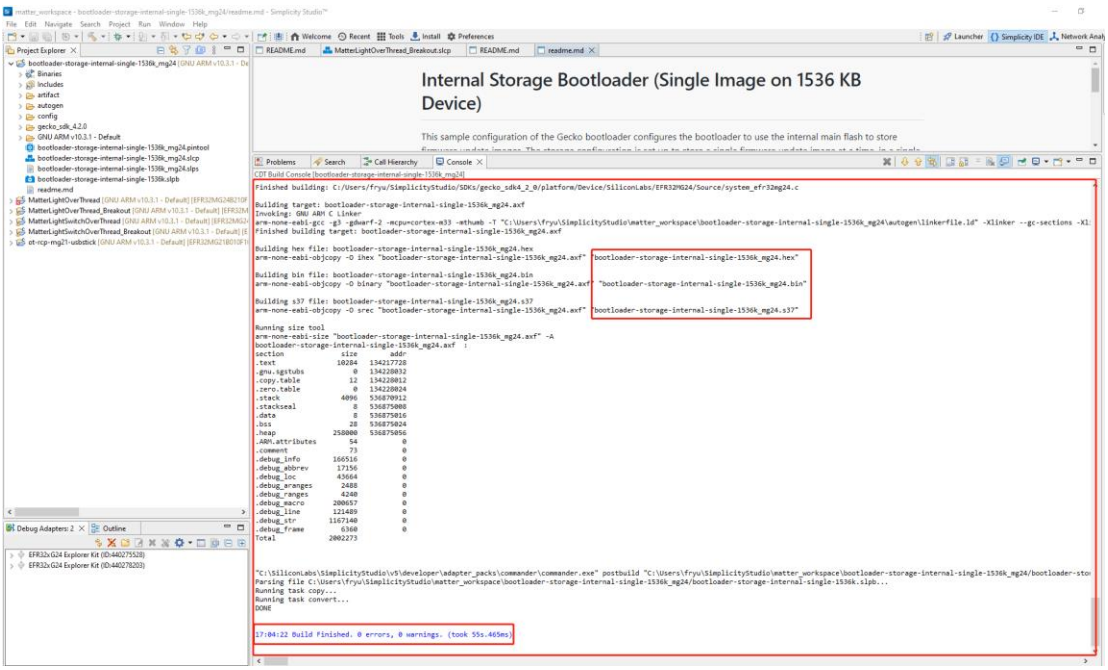
3. 工程创建完成后，如下：



4. 编译工程。选择工程，点击`锤子图标`进行编译。



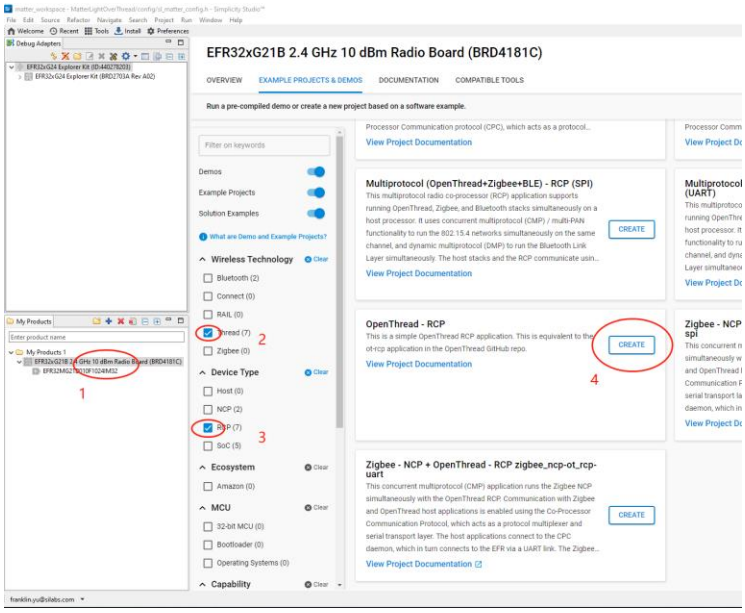
5. 编译完成后如下图所示。可看到编译出的固件（.s37/.bin/.hex）和编译时间。



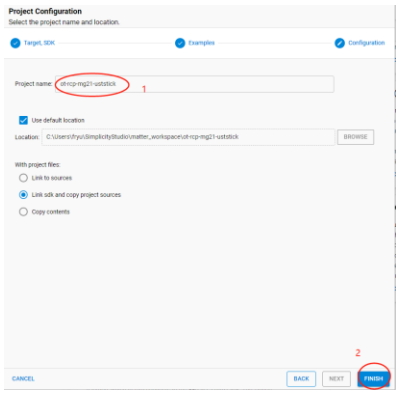
5.4 基于 GSDK 开发 OT-RCP

5.4.1 创建&编译工程

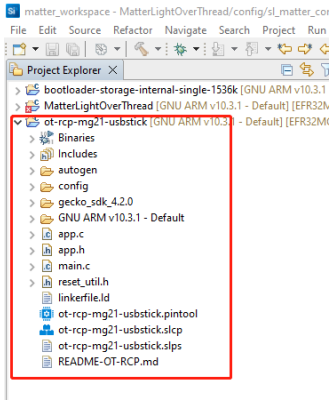
1. 选择`BRD4181C`作为基础，创建`OpenThread-RCP`工程。



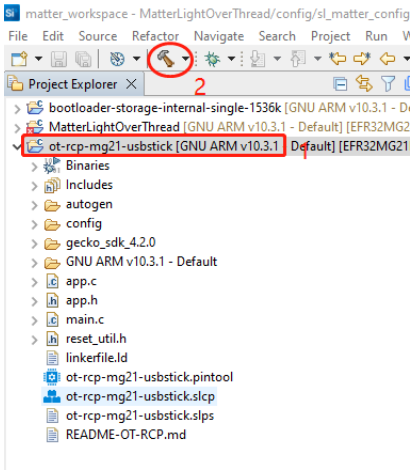
2. 修改工程名为`ot-rcp-mg21-uststick`并创建。（改工程名不是必须的）



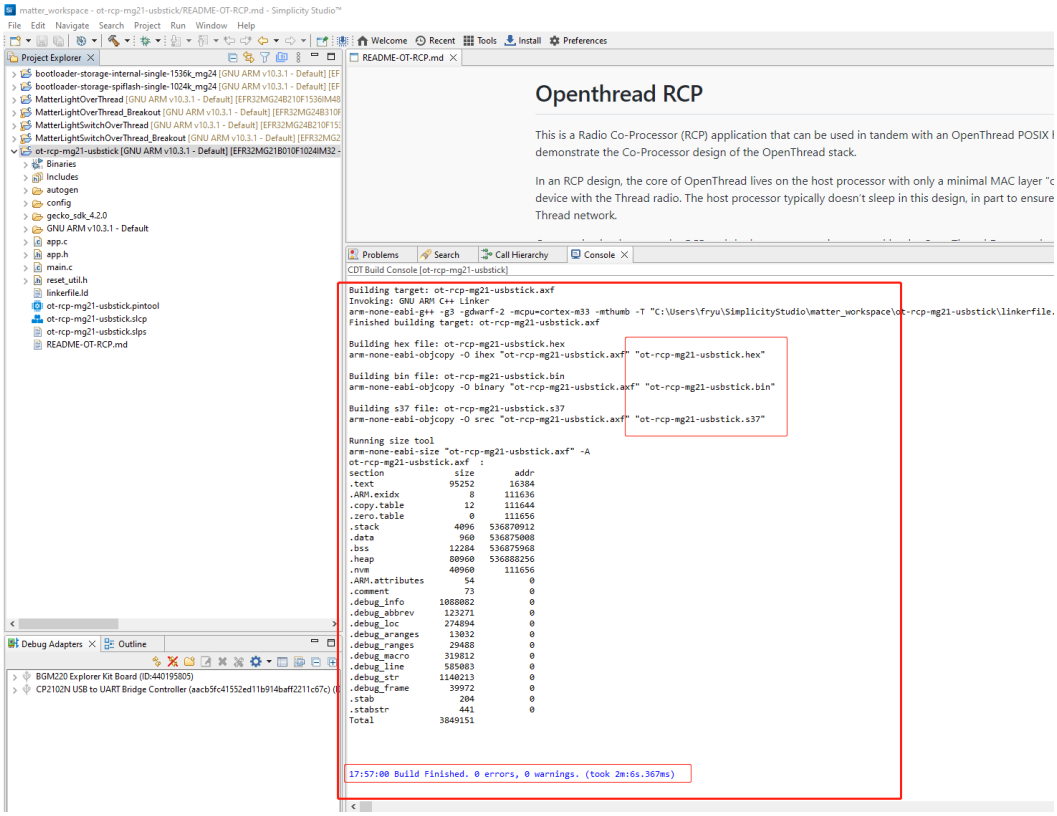
3. 工程创建完成后可看到如下图所示：



4. 编译工程。选择工程，点击`锤子图标`进行编译。



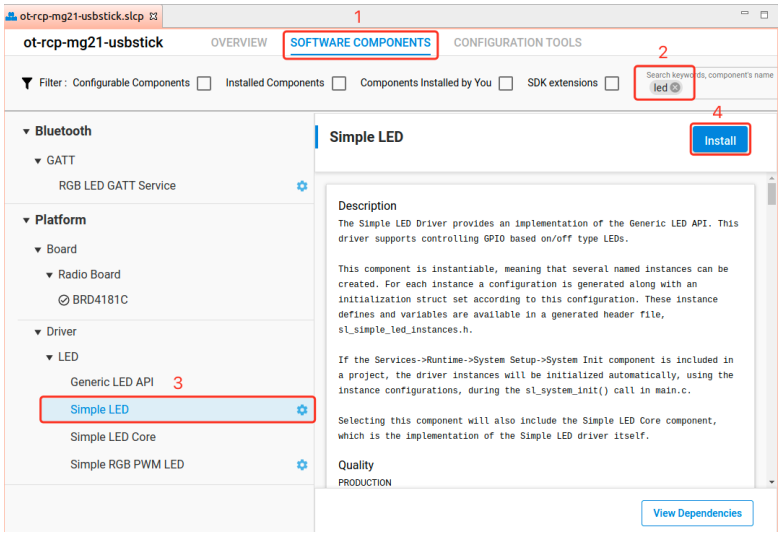
5. 编译完成后如下图所示。可看到编译出的固件（.s37/.bin/.hex）和编译时间。



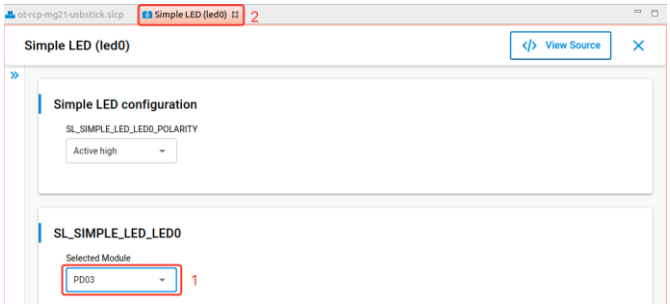
5.4.2 修改工程

5.4.2.1 添加 LED 控制组件

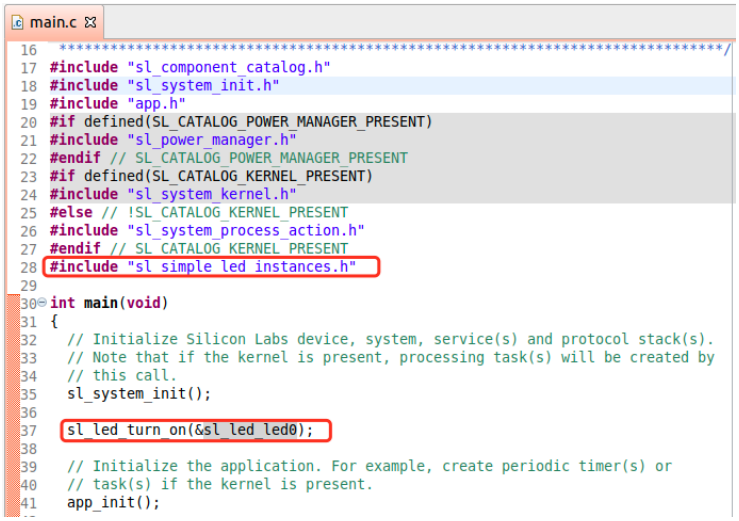
1. 打开`ot-rcp-mg21-usbstick.slcp`安装 LED 软件控制组件



2. 选择 LED 对应的 GPIO 引脚：“PD03”并关闭配置

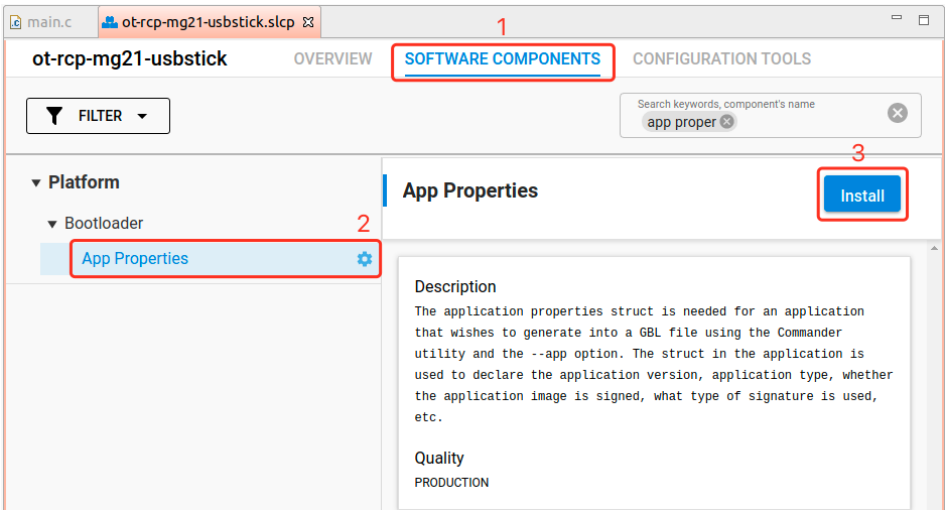


3. 代码中添加 LED 的控制



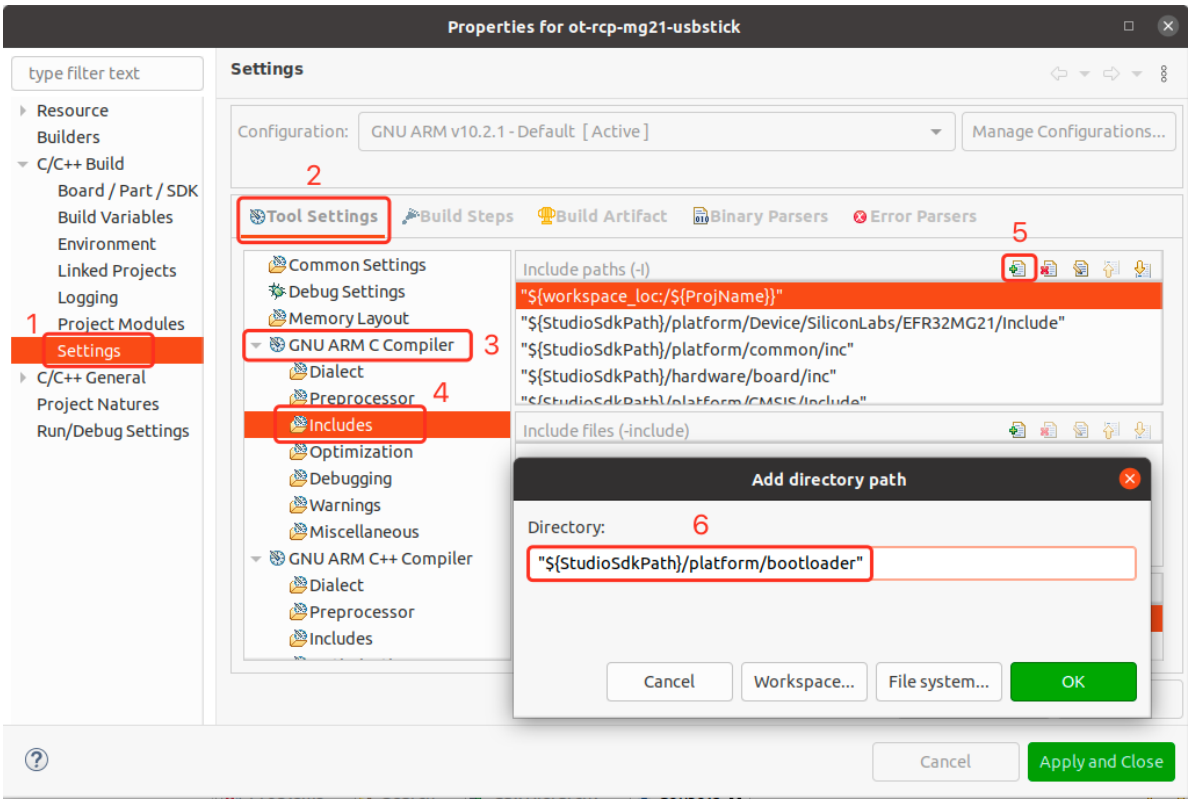
5.4.2.2 添加 x-modem bootloader 功能

1. 安装组件：`Platform->Bootloader->App Properties`



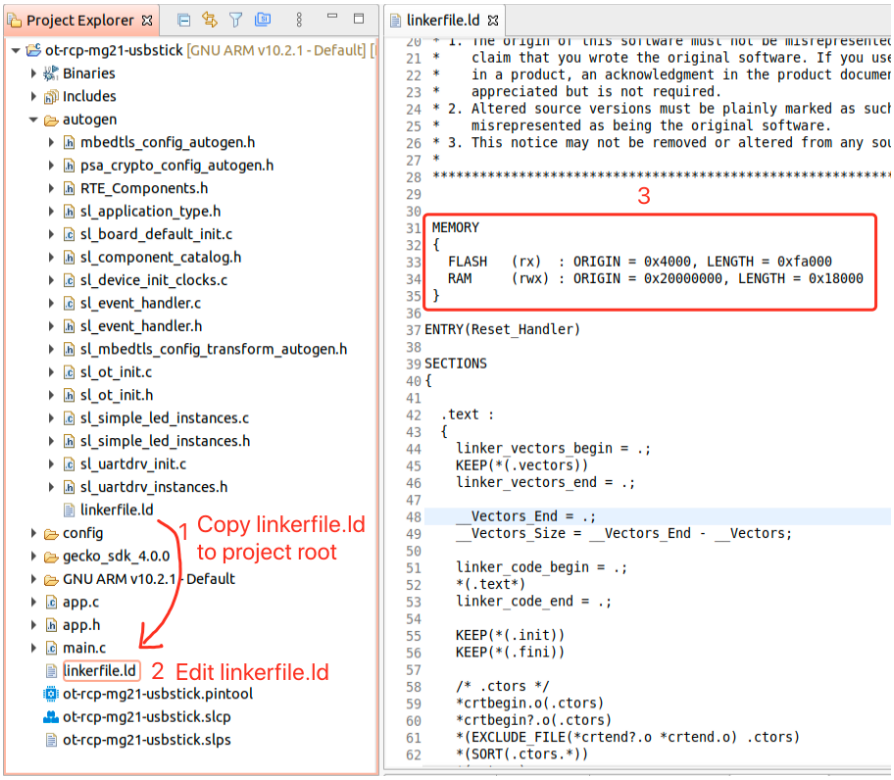
2. 增加头文件

右键工程选择`Properties`，在弹出的对话框中选择`【C/C++ Build】->【Settings】->【Tool Setttings】->【GNU ARM C Compilier】->【Includes】`，添加一个目录`${StudioSdkPath}/platform/bootloader`。如图所示：

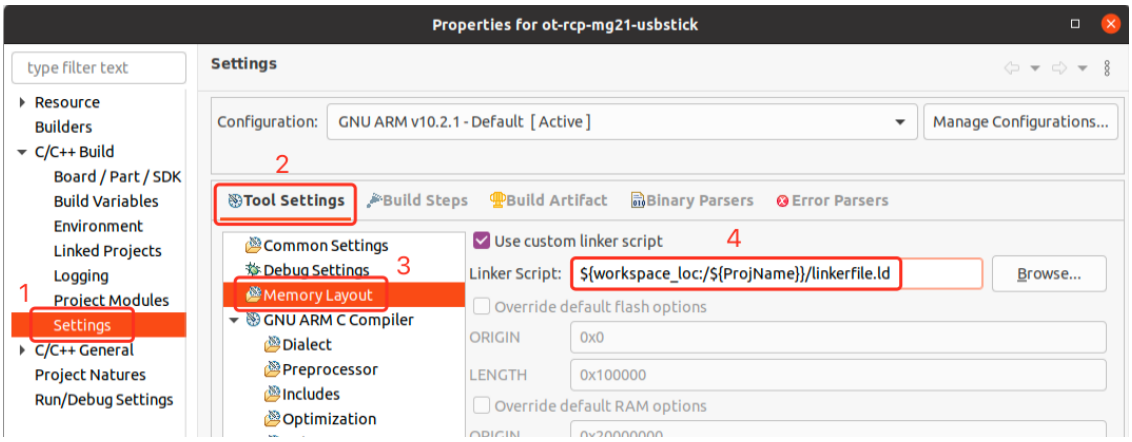


5.4.2.3 为 bootloader 功能修改连接脚本

1. 拷贝文件`autogen/linkerfile.ld`粘贴到项目根目录下，并按下图修改`linkerfile.ld`内容。



2. 在项目属性对话框内，引用新的`Linker Script`路径，指向根目录下的`linkfile.ld`文件如下图



5.4.2.4 重新编译工程

参考前面的编译步骤，编译完成后可在编译目录中找到固件：`ot-rcp-mg21-usbstick.s37`

5.4.2.5 转换成 GBL 升级文件

使用`commander`工具将固件`ot-rcp-mg21-usbstick.s37`转换成`ot-rcp-mg21-usbstick.gbl`格式的升级文件

```
C:\ot-rcp-mg21-usbstick\GNU ARM v10.3.1 - Default>commander gbl create ot-rcp-mg21-usbstick.gbl --
app ot-rcp-mg21-usbstick.s37
Parsing file ot-rcp-mg21-usbstick.s37...
Initializing GBL file...
Adding application to GBL...
Writing GBL file ot-rcp-mg21-usbstick.gbl...
DONE
```

利用`fw_upgrade_utility`工具将`ot-rcp-mg21-usbstick.gbl`文件升级到 USB Stick 中

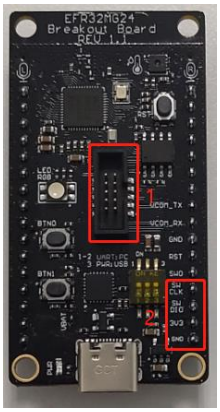
5.5 烧录设备固件

通过前面的步骤，目前已经编译出以下固件：

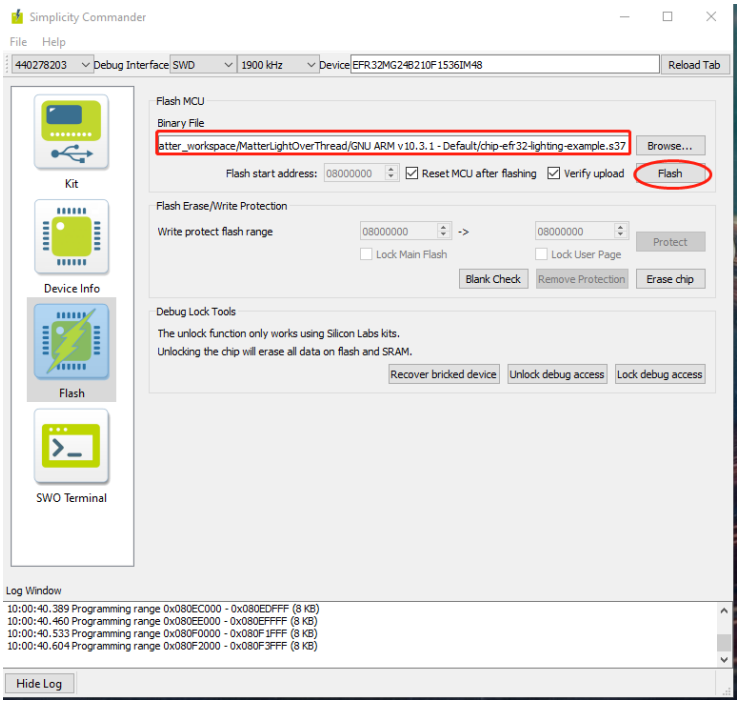
固件	目标板	说明
MatterLightOverThread_Breakout.s37	EFR32MG24 Breakout Board REV 1.1	Matter over Thread 的灯应用固件
MatterLightSwitchOver-Thread_Breakout.s37	EFR32MG24 Breakout Board REV 1.1	Matter over Thread 的开关应用固件
ot-rcp-mg21-usbstick.gbl	EFR32MG21 USB STICK REV 2.0	Thread 边界路由器（RCP 模式)
bootloader-storage-internal-single-1536k_mg24.s37	EFR32MG24 Breakout Board REV 1.1	EFR32MG24 开发板通用 bootloader

1．使用`Simplicity Commander`分别给两块 Breakout Board 烧录固件。

- JLink 硬件连接：图中两个接口二选一。



- 连接好硬件后，使用 Simplicity Commander 工具进行烧录。烧录方式参考下图：



注意事项：

a) Matter 灯需要同时烧录`MatterLightOverThread_Breakout.s37`和`bootloader-storage-internal-single-1536k_mg24.s37`

b) Matter 开关需要同时烧录`MatterLightSwitchOverThread_Breakout.s37`和`bootloader-storage-internal-single-1536k_mg24.s37`

2．在 ubuntu 环境中升级 USB Stick 固件

- 将 USB Stick 插入 PC，并查看其端口

```
$ ls /dev/ttyUSB*
ttyUSB0
```

- 停止`otbr-agent`以释放`tty`端口

```
$ sudo systemctl stop otbr-agent.service
```

- 升级固件到 USB Stick 中

```
$ cd usbstick/fw_upgrade_util/
$ sudo ./fw_upgrade_utility -f ~/Desktop/firmware/ot-rcp-mg21-usbstick.gbl -p /dev/ttyUSB0
```

- 升级完成后，再重新启动 otbr-agent

```
$ sudo systemctl start otbr-agent.service
```

5.6 通过终端命令行控制 Matter 灯

设备入网控制操作步骤

- 运行 otbr-agent
- 使用 ot-ctl 创建 thread 网络
- 使用 chip-tool 配置设备入网
- 使用 chip-tool 控制设备

5.6.1 运行 otbr-agent

1. 将`USB Stick`插入电脑的`USB`端口
2. 编辑文件`/etc/default/otbr-agent`

```
$ sudo vi /etc/default/otbr-agent

# 虚拟机(网络端口为 enp0s3, USB Stick 端口为/dev/ttyUSB0)
OTBR_AGENT_OPTS="-I wpan0 -B enp0s3 spinel+hdlc+uart:///dev/ttyUSB0?uart-baudrate=115200"

# 树莓派
OTBR_AGENT_OPTS="-I wpan0 -B eth0 spinel+hdlc+uart:///dev/ttyUSB0?uart-baudrate=115200"
```

3. 启动`otbr-agent`服务

```
$ sudo systemctl start otbr-agent.service
```

后续步骤中的指令仅作为参考，不是必须的。

4. 检查：`/var/log/syslog`输出`otbr-agent`的运行日志

```
$ tail -f /var/log/syslog
```

5. 检查主机与 RCP 的连接状态

```
$ sudo ot-ctl state
# 如果连接失败，会返回以下信息
connect session failed: No such file or directory
```

6. 用`ot-ctl CLI`命令检查`ot-br-posix`的版本号

```
$ sudo ot-ctl version
OPENTHREAD/3abe1693d-dirty; POSIX; Dec  7 2022 15:58:36
Done
```

7. 用`ot-ctl CLI`命令检查`ot-rcp`的固件版本号

```
$ sudo ot-ctl rcp version
SL-OPENTHREAD/2.2.0.0_GitHub-91fa1f455; EFR32; Jan  5 2023 18:43:40
Done
```

8. 停止`otbr-agent`服务

```
$ sudo systemctl stop otbr-agent.service
```

5.6.2 使用 ot-ctl 创建 thread 网络

通过`ot-ctl`创建 Thread 网络，查看网络配置。

1. 创建 Thread 网络

```
$ sudo ot-ctl dataset init new
Done
$ sudo ot-ctl dataset networkkey 00112233445566778899aabbccddeeff
Done
$ sudo ot-ctl dataset extpanid 1111111122222222
Done
$ sudo ot-ctl dataset panid 0x1234
Done
$ sudo ot-ctl dataset channel 15
Done
# 将以上配置提交为活动配置
$ sudo ot-ctl dataset commit active
Done
# 打开 ipv6 接口
$ sudo ot-ctl ifconfig up
Done
# 启动 Thread 协议
$ sudo ot-ctl thread start
Done
```

2. 查看 Thread 网络配置

```
$ sudo ot-ctl dataset active -x
0e080000000000010000000300000f35060004001fffe00208111111112222222220708fdb0ab694c9b3a17051000112233445566778899aabbccddeeff030f4f70656e5468726561642d663663610102123404 10d237761823728dd2cbfe64f477b38b4c0c0402a0f7f8
Done
```

5.6.3 使用 chip-tool 配置设备入网

1. 配置 Matter 灯入网

```
$ ./chip-tool pairing ble-thread ${NODE_ID} hex:${DATASET} ${PIN_CODE} ${DISCRIMINATOR}
```

- ${NODE_ID}：给入网设备分配一个 ID。可以是 RCP 初始化之后，未使用过的任何非零值，chip-tool 使用它来操作这个 Matter 设备。
- ${DATASET}：通过指令`sudo ot-ctl dataset active -x`获取
- ${PIN_CODE}：如果尚未在闪存中配置，请使用默认配对代码。参考头文件：`CHIPProjectConfig.h`
- ${DISCRIMINATOR}：如果尚未在闪存中配置，请使用默认配对代码。参考头文件：`CHIPProjectConfig.h`

```
// Use a default pairing code if one hasn't been provisioned in flash.
#ifndef CHIP_DEVICE_CONFIG_USE_TEST_SETUP_PIN_CODE
#define CHIP_DEVICE_CONFIG_USE_TEST_SETUP_PIN_CODE 20202021
#endif

#ifndef CHIP_DEVICE_CONFIG_USE_TEST_SETUP_DISCRIMINATOR
#define CHIP_DEVICE_CONFIG_USE_TEST_SETUP_DISCRIMINATOR 0xF00  //转为十进制数为 3840
#endif
```

2. 给 Matter 灯设备上电。长按 BTN0 6 秒进入配对状态。
3. 分配`1001`作为 Matter 灯的 NODE_ID。将 Matter 灯加入网络的指令如下：

```
$ sudo ./chip-tool pairing ble-thread 1001
hex:0e080000000000010000000300000f35060004001fffe0020811111111222222220708fd67d3ca68dbeac6051000112233445
566778899aabbccddeeff030f4f70656e5468726561642d3065376401021234041 0b58c67a8a3aaa68557be489b35798ad60c0402
a0f7f8 20202021 3840

# 上面指令运行正常的话，最后会看到类似这样的信息
[1673925106.632759][2792:2792] CHIP:DL: Inet Layer shutdown
[1673925106.632762][2792:2792] CHIP:DL: BLE shutdown
[1673925106.632764][2792:2792] CHIP:DL: System Layer shutdown
```

5.6.4 使用 chip-tool 控制 Matter 灯

1. 切换灯状态

```
$ sudo ./chip-tool onoff toggle ${NODE_ID} 1

# 实例
$ sudo ./chip-tool onoff toggle 1001 1
```

2. 打开灯

```
$ sudo ./chip-tool onoff on ${NODE_ID} 1

# 实例
$ sudo ./chip-tool onoff on 1001 1
```

3. 关闭灯

```
$ sudo ./chip-tool onoff off ${NODE_ID} 1

# 实例
$ sudo ./chip-tool onoff off 1001 1
```

4. 读取灯的状态

```
$ sudo ./chip-tool onoff read on-off ${NODE_ID} 1

# 实例
$ sudo ./chip-tool onoff read on-off 1001 1
```

5.7 使用 Matter 开关控制 Matter 灯

1. 给 Matter 开关设备上电。长按 BTN0 6 秒进入配对状态。
2. 分配`1002`作为 Matter 开关的 NODE_ID。将 Matter 开关加入网络的指令如下：

```
$ sudo ./chip-tool pairing ble-thread 1002
hex:0e080000000000010000000300000f35060004001fffe0020811111111222222220708fd67d3ca68dbeac605100011223344556
6778899aabbccddeeff030f4f70656e5468726561642d3065376401021234041 0b58c67a8a3aaa68557be489b35798ad60c0402a0f7
f8 20202021 3840
```

3. 设置灯的 ACL，让开关可以控制它

   ACL 参数说明请参阅 Matter Core Specification `9.10.5.3. ACL Attribute`

```
$ sudo ./chip-tool accesscontrol write acl '[{"fabricIndex":1, "privilege":5, "authMode":2, "sub-
jects":[112233, ${switch_node_id}], "targets":null}]' ${lighting_node_id} 0
```

   实例：

```
$ sudo ./chip-tool accesscontrol write acl '[{"fabricIndex":1, "privilege":5, "authMode":2, "sub-
jects":[112233, 1002], "targets":null}]' 1001 0

# 上面指令运行正常的话，最后会看到类似这样的信息
[1673925106.632759][2792:2792] CHIP:DL: Inet Layer shutdown
[1673925106.632762][2792:2792] CHIP:DL: BLE shutdown
[1673925106.632764][2792:2792] CHIP:DL: System Layer shutdown
```

4. 让开关绑定灯

```
$ sudo ./chip-tool binding write binding '[{"fabricIndex":1, "node":${lighting_node_id}, "endpoint":1,
"cluster":6}]' ${switch_node_id} 1
```

   实例：

```
$ sudo ./chip-tool binding write binding '[{"fabricIndex":1, "node":1001, "endpoint":1, "cluster":6}]' 1002
1

# 上面指令运行正常的话，最后会看到类似这样的信息
[1673925106.632759][2792:2792] CHIP:DL: Inet Layer shutdown
[1673925106.632762][2792:2792] CHIP:DL: BLE shutdown
[1673925106.632764][2792:2792] CHIP:DL: System Layer shutdown
```

完成以上操作后，就可以使用 Matter 开关来控制 Matter 灯的亮灭了。

6 常见问题

- ubuntu 中找不到 OTBR 和 BLE Dongle
  - 请确认是否有将这些设备添加到 ubuntu 的 USB 设备中
- BLE Dongle 连接确认

  已测试过的型号：ORICO/奥睿科蓝牙 5.0 适配器（可在淘宝商城购买到）

  连接正常与否，可参考以下指令进行检测：

```
# 使用 hciconfig 显示 hci 设备列表
$ sudo hciconfig -a
  hci0:   Type: Primary  Bus: USB
          BD Address: 8C:88:2B:66:40:83  ACL MTU: 1021:6  SCO MTU: 255:12
          UP RUNNING
          RX bytes:1611 acl:0 sco:0 events:176 errors:0
          TX bytes:31970 acl:0 sco:0 commands:176 errors:0
          Features: 0xff 0xff 0xff 0xfe 0xdb 0xfd 0x7b 0x87
          Packet type: DM1 DM3 DM5 DH1 DH3 DH5 HV1 HV2 HV3
          Link policy: RSWITCH HOLD SNIFF PARK
          Link mode: PERIPHERAL ACCEPT
          Name: 'ubuntu-VM'
          Class: 0x6c0000
          Service Classes: Rendering, Capturing, Audio, Telephony
          Device Class: Miscellaneous,
          HCI Version: 5.1 (0xa)  Revision: 0x9a9
          LMP Version: 5.1 (0xa)  Subversion: 0x8a6b
          Manufacturer: Realtek Semiconductor Corporation (93)

# 使用 hcitool 扫描附近 BLE 设备
$ sudo hcitool lescan
LE Scan ...
07:71:C5:CC:97:05 (unknown)
16:D1:F6:B4:E7:B9 (unknown)
3D:68:5F:85:ED:FA (unknown)
```

- 关于 linux 应用程序的编译及获取：参考`Matter 开发：基于 Linux 的应用程序.pdf`

7 参考资料

- 芯科科技 Simplicity-studio 集成开发环境：https://www.silabs.com/developers/simplicity-studio
- 芯科科技开发者文档：https://docs.silabs.com/
- 芯科科技 Matter 方案介绍：https://www.silabs.com/wireless/matter
- 芯科科技 Matter 开发文档：https://docs.silabs.com/matter/1.0.1/matter-start/
- Matter 协议规格书：https://csa-iot.org/developer-resource/specifications-download-request/
- OpenThread 参考资料：https://openthread.google.cn/

8 文档修订历史

Revision 1.0.0

Jan 31, 2023

- 初始版本

- 文档结束 -