



CHIP - Connected Home Over IP

This lab procedure walks through the steps to build CHIP Lock-app project using EFR32 SLWRB4170A. The first part reviews how to setup the environment using Virtual Machine running Ubuntu 20.04 LTS on Windows 10 machine. The second part shows how to create Lock-app example and flash on to SLWRB4170A. The final part introduces how to establish the connection with OTBR and use Chip-tool to control the device.

*As of 5/11/2021 Project Connected Home over IP is now Matter. Learn more about [Matter](#).

KEY POINTS

- Setup CHIP environment on WSL2
- Build Lock-app project for BRD4170A
- Flash firmware on BRD4170A
- Establish communication with OTBR
- Use chip-tool to control the device

Prerequisites

1 Prerequisites

For this lab you will need the following:

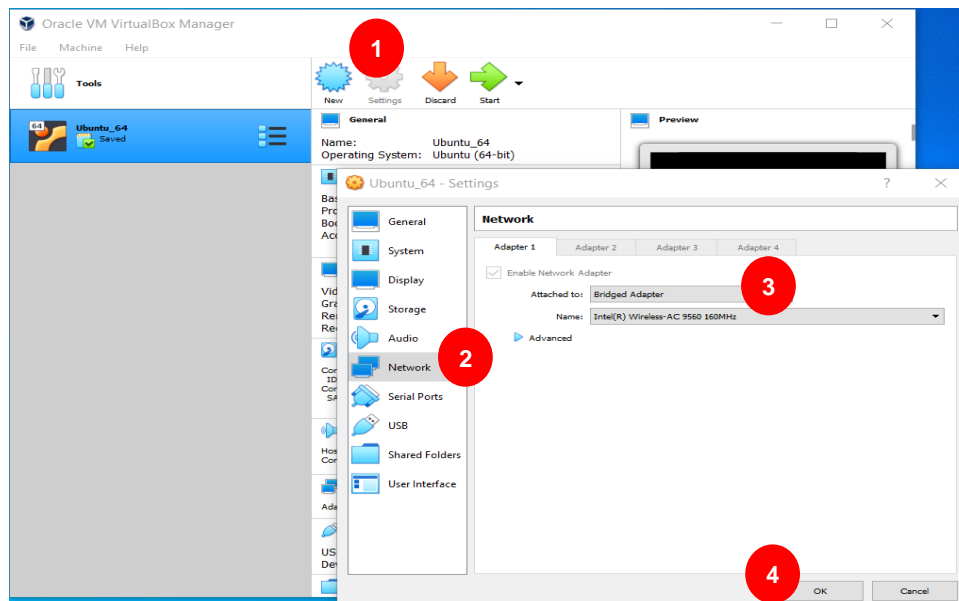
- Software Requirement:
 1. Studio 5 with Gecko SDK 3.0.0 or later installed on Windows 10
 2. VirtualBox (with **Ubuntu 20.04 LTS**) installed on Windows 10
 - VirtualBox: <https://www.virtualbox.org/>
 - Ubuntu **20.04.x** LTS: <https://ubuntu.com/download/desktop>
 3. Latest Raspberry Pi OS.
 - For more instructions visit: <https://www.raspberrypi.org/documentation/installation/noobs.md>
 4. Gecko SDK v2.7 downloaded in Linux (Ubuntu) VM
 - Download from https://github.com/SiliconLabs/sdk_support
 5. Tera Term: <https://tssh2.osdn.jp/index.html.en>
 6. J-Link RTT Viewer: <https://www.segger.com/downloads/jlink/>
 7. SSH Client (Putty or Similar) : <https://www.putty.org/>

Note: WSL2 or Native Linux or Mac machine can also be used to build this CHIP Project but WSL2 has limitations as described in [section 5.4](#). To install WSL2 visit: <https://docs.microsoft.com/en-us/windows/wsl/install-win10>

- Hardware Requirements:
 1. SLWSTK6000B Wireless Starter Kit main board + BRD4170A for CHIP device
 2. SLWSTK6000B Wireless Starter Kit main board + BRD4170A for RCP Device
 3. Raspberry Pi 3B+/4 for Open Thread Border Router
 4. Two Micro-USB to USB Type-A cables for Kit

1.1 Prepare Network setting for VirtualBox

Assuming Ubuntu is installed on VM in Windows 10. Before running the VM, we need to configure networking setting and change network adapter to Bridge mode.



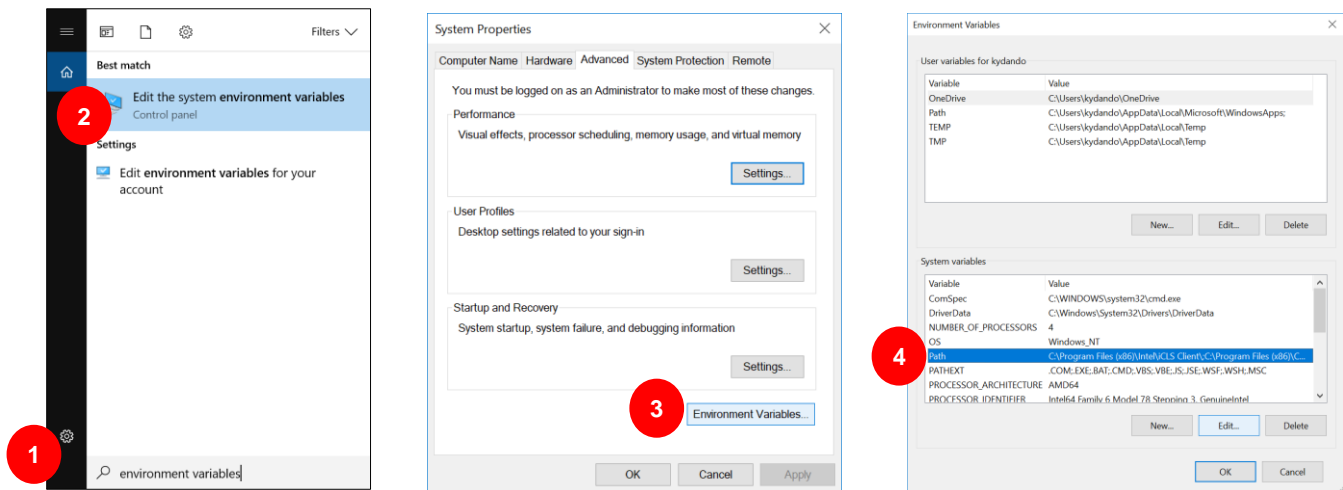
Prerequisites

1. Open the VirtualBox and click on **'Settings'** of your Linux VM.
2. Click on **'Network'** and Select the **'Adapter 1'** tab.
3. Make sure **'Bridge Adapter'** is selected in the **'Attached to:'** box.
4. Click OK.

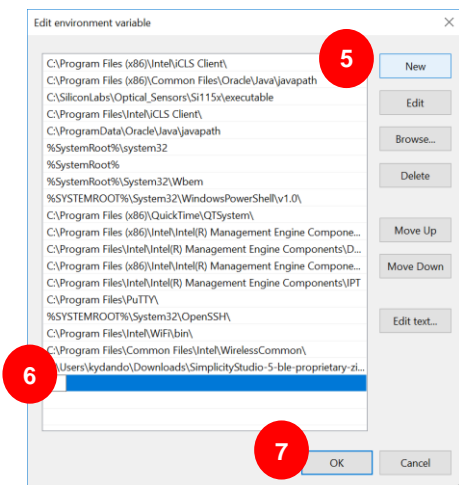
1.2 Prepare Windows 10 OS to run Simplicity Commander (optional)

For simplicity's sake add the path of Simplicity Commander (`C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander`) to the `path` environment variable in your OS, so that the `commander.exe` can be called from any directory as well as from Linux terminal.

1. Use Search bar in Windows 10 to locate **"Environment Variables"** menu.
2. Select **"Edit the system environment variables"**.



3. Select **"Environment Variables"** in the System Properties window.
4. Select **"Path"** under System Variables. Click **"Edit..."**



5. Click **"New"**.
6. Verify this is the valid path for your installation. **`C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander`**
7. Most default installations will have this path. It will be different if you modified the installation folder for Simplicity Studio.
8. Paste the verified location of Simplicity Commander

Prerequisites

9. Click **“OK”** to accept addition to path and return to Environment Variables window
10. Under System variables Click **“New”**
11. Under Variable name enter **“PATH_GCCARM”**
12. Under Variable value enter **“C:\SiliconLabs\SimplicityStudio\v5\developer\toolchains\gnu_arm7.2_2017q4”**
13. Click **“OK”** to accept new variable.
14. Under System variable Click **“New”**
15. Under Variable name enter **“PATH_SCMD”**
16. Under Variable value enter **“C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander”** (VERIFIED IN STEP 5)
17. Click **“OK”** to accept new variable.
18. Click **“OK”** 3 times to close the open windows
19. Computer will need to be restarted for new PATH variable to be updated

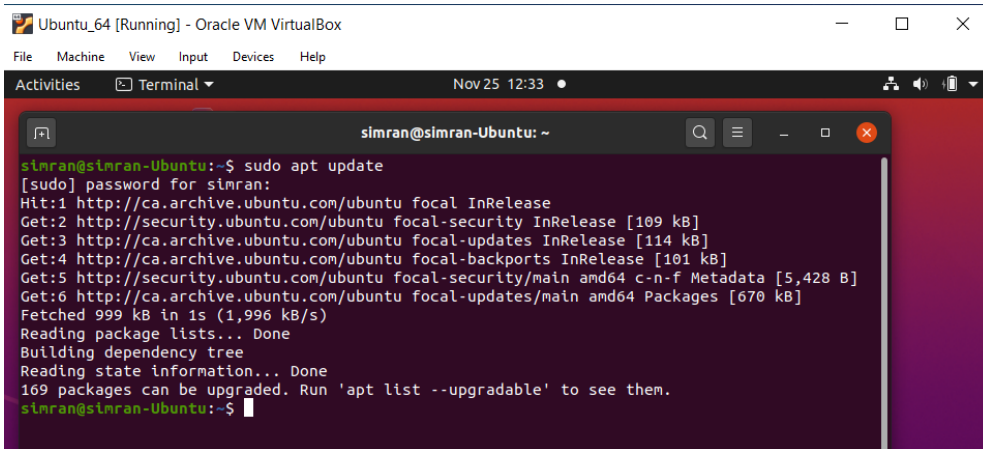
1.3 Prepare Linux packages

Note: Following instructions can also work on WSL2 (Windows 10), Native Linux or Mac machine

In Virtualbox, start **Ubuntu 20.04 VM** and open terminal window.

Update the latest packages by typing following commands in terminal window:

1 sudo apt update

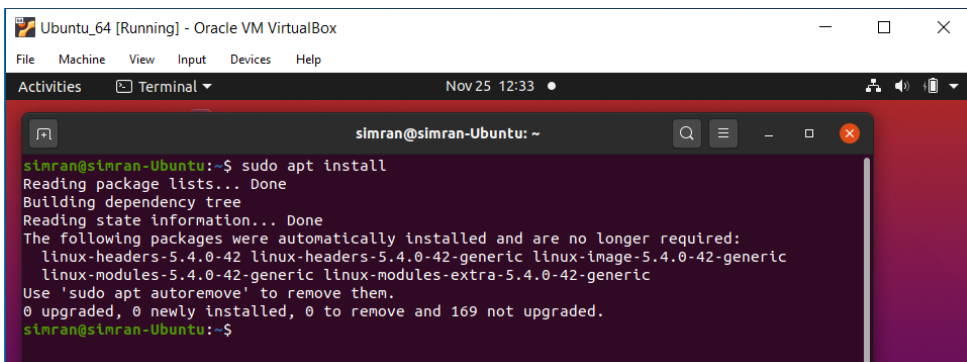


```

simran@simran-Ubuntu: ~
simran@simran-Ubuntu:~$ sudo apt update
[sudo] password for simran:
Hit:1 http://ca.archive.ubuntu.com/ubuntu focal InRelease
Get:2 http://security.ubuntu.com/ubuntu focal-security InRelease [109 kB]
Get:3 http://ca.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://ca.archive.ubuntu.com/ubuntu focal-backports InRelease [101 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64 c-n-f Metadata [5,428 B]
Get:6 http://ca.archive.ubuntu.com/ubuntu focal-updates/main amd64 Packages [670 kB]
Fetched 999 kB in 1s (1,996 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
169 packages can be upgraded. Run 'apt list --upgradable' to see them.
simran@simran-Ubuntu:~$

```

2 sudo apt install



```

simran@simran-Ubuntu: ~
simran@simran-Ubuntu:~$ sudo apt install
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-5.4.0-42 linux-headers-5.4.0-42-generic linux-image-5.4.0-42-generic
  linux-modules-5.4.0-42-generic linux-modules-extra-5.4.0-42-generic
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 169 not upgraded.
simran@simran-Ubuntu:~$

```

2 Build Preparation for CHIP

Note: Instructions provided in this **section 2** can be used to build Chip project on WSL2 (Windows 10), Native Linux or Mac machine)

2.1 Prerequisites for CHIP project on Linux VM

The following section details how to set up environment for CHIP Project. All the current development on CHIP is available at <https://github.com/project-chip/connectedhomeip>

Check out the Code and SDK from GitHub.

1. Open the Linux terminal from start menu.
2. Clone the Gecko SDK 2.7 from https://github.com/SiliconLabs/sdk_support

```
git clone https://github.com/SiliconLabs/sdk_support.git
```

The screenshot shows a terminal window titled "simran@simran-Ubuntu: ~" with the following output:

```
simran@simran-Ubuntu:~$ git clone https://github.com/SiliconLabs/sdk_support.git
Cloning into 'sdk_support'...
remote: Enumerating objects: 21, done.
remote: Counting objects: 100% (21/21), done.
remote: Compressing objects: 100% (15/15), done.
remote: Total 24707 (delta 7), reused 0 (delta 0), pack-reused 24686
Receiving objects: 100% (24707/24707), 104.89 MiB | 27.92 MiB/s, done.
Resolving deltas: 100% (14859/14859), done.
Updating files: 100% (28981/28981), done.
simran@simran-Ubuntu:~$
```

3. Clone the repository. To check out the CHIP repository:

```
ls
```

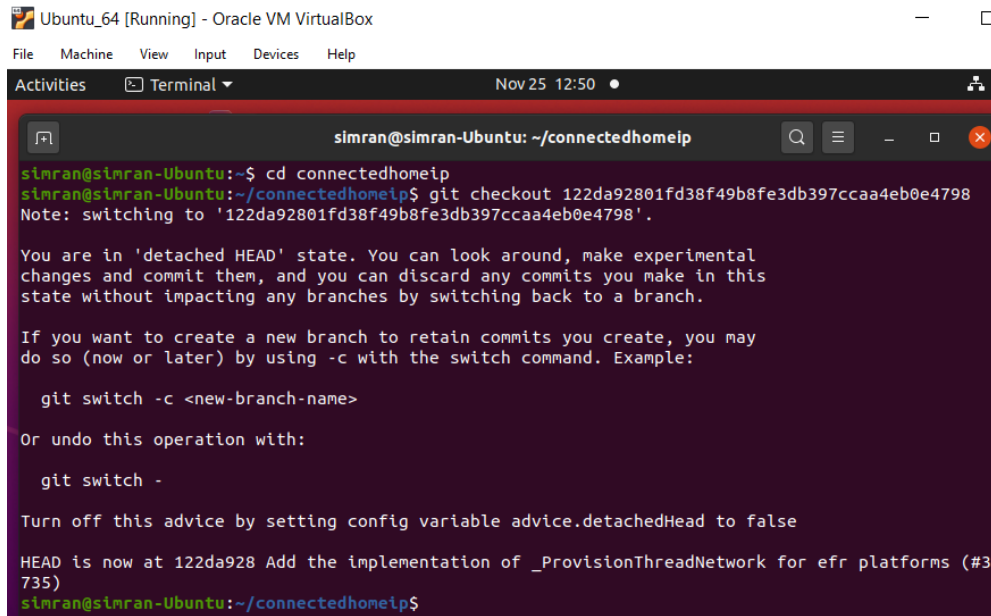
The screenshot shows a terminal window titled "simran@simran-Ubuntu: ~" with the following output:

```
simran@simran-Ubuntu:~$ git clone https://github.com/project-chip/connectedhomeip.git
Cloning into 'connectedhomeip'...
remote: Enumerating objects: 30370, done.
remote: Total 30370 (delta 0), reused 0 (delta 0), pack-reused 30370
Receiving objects: 100% (30370/30370), 13.09 MiB | 29.39 MiB/s, done.
Resolving deltas: 100% (21458/21458), done.
simran@simran-Ubuntu:~$
```

Build Preparation for CHIP

4. Switch to **'connectedhomeip'** directory and checkout the commit [122da92801fd38f49b8fe3db397ccaa4eb0e4798](#) (At the time of this tutorial, following commit hash is tested. Latest commit on GitHub should work as well)

```
git checkout 122da92801fd38f49b8fe3db397ccaa4eb0e4798
```



The screenshot shows a terminal window titled 'simran@simran-Ubuntu: ~/connectedhomeip'. The user has executed the command `git checkout 122da92801fd38f49b8fe3db397ccaa4eb0e4798`. The output indicates a successful checkout to a detached HEAD state. The terminal text is as follows:

```
simran@simran-Ubuntu:~$ cd connectedhomeip
simran@simran-Ubuntu:~/connectedhomeip$ git checkout 122da92801fd38f49b8fe3db397ccaa4eb0e4798
Note: switching to '122da92801fd38f49b8fe3db397ccaa4eb0e4798'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

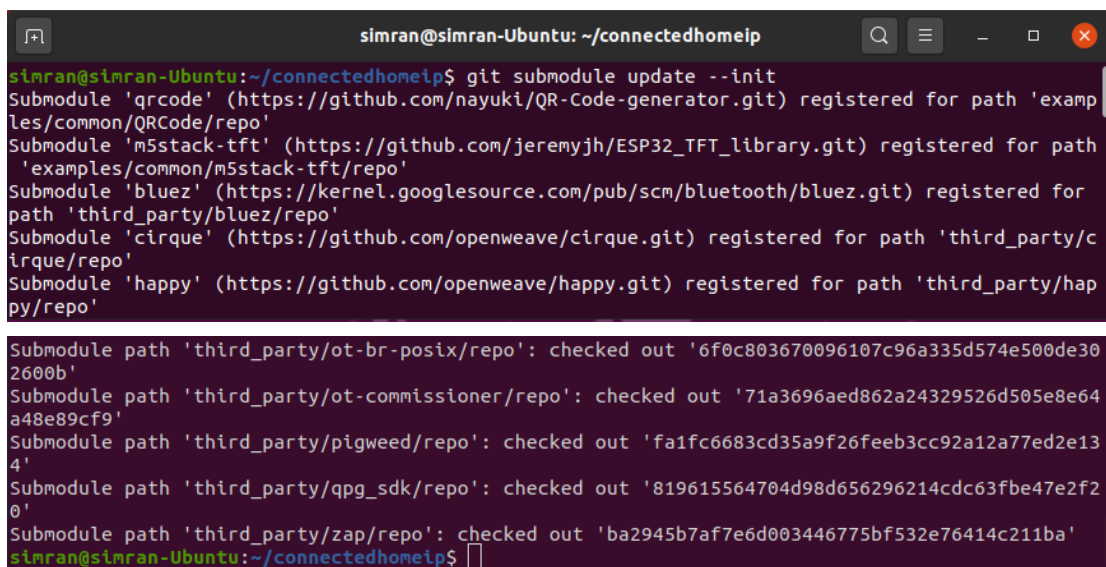
  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 122da928 Add the implementation of _ProvisionThreadNetwork for efr platforms (#3
735)
simran@simran-Ubuntu:~/connectedhomeip$
```

5. If you already have a checkout, run the following command to sync submodules:

```
git submodule update --init
```



The screenshot shows a terminal window titled 'simran@simran-Ubuntu: ~/connectedhomeip'. The user has executed the command `git submodule update --init`. The output lists several submodules and their checked-out commit hashes. The terminal text is as follows:

```
simran@simran-Ubuntu:~/connectedhomeip$ git submodule update --init
Submodule 'qrcode' (https://github.com/nayuki/QR-Code-generator.git) registered for path 'examples/common/QRCode/repo'
Submodule 'm5stack-tft' (https://github.com/jeremyjh/ESP32_TFT_library.git) registered for path 'examples/common/m5stack-tft/repo'
Submodule 'bluez' (https://kernel.googlesource.com/pub/scm/bluetooth/bluez.git) registered for path 'third_party/bluez/repo'
Submodule 'cirque' (https://github.com/openweave/cirque.git) registered for path 'third_party/cirque/repo'
Submodule 'happy' (https://github.com/openweave/happy.git) registered for path 'third_party/happy/repo'

Submodule path 'third_party/ot-br-posix/repo': checked out '6f0c803670096107c96a335d574e500de302600b'
Submodule path 'third_party/ot-commissioner/repo': checked out '71a3696aed862a24329526d505e8e64a48e89cf9'
Submodule path 'third_party/pigweed/repo': checked out 'fa1fc6683cd35a9f26feeb3cc92a12a77ed2e134'
Submodule path 'third_party/qpg_sdk/repo': checked out '819615564704d98d656296214cdc63fbc47e2f20'
Submodule path 'third_party/zap/repo': checked out 'ba2945b7af7e6d003446775bf532e76414c211ba'
simran@simran-Ubuntu:~/connectedhomeip$
```

6. Install prerequisites on Linux required to CHIP project.

```
sudo apt-get install git gcc g++ python pkg-config libssl-dev libdbus-1-dev libglib2.0-dev libavahi-client-dev ninja-build python3-venv python3-dev unzip
```

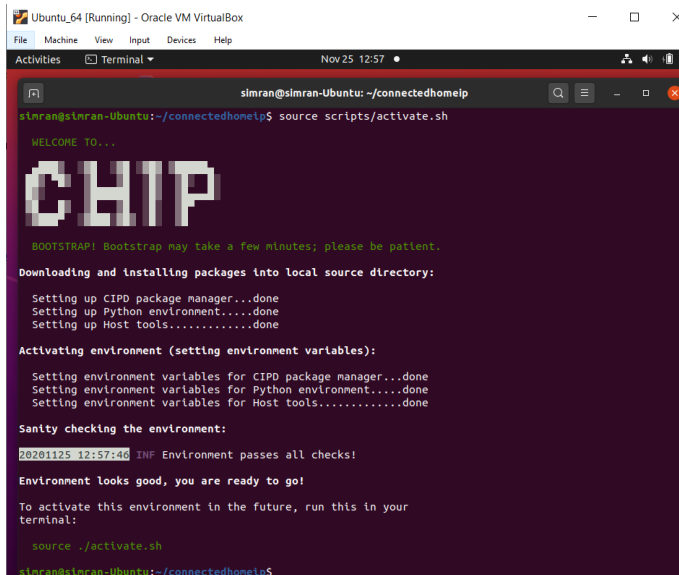
2.2 Prepare CHIP Environment and build the source and libraries

Build all sources, libraries, and tests for the host platform.

1. Activate the environment builds

```
source scripts/activate.sh
```

Before running any other build command, the environment setup script should be sourced at the top level. This script takes care of downloading GN, ninja, and setting up a Python environment with libraries used to build and test.



```
simran@simran-Ubuntu: ~/connectedhomeip
simran@simran-Ubuntu:~/connectedhomeip$ source scripts/activate.sh
WELCOME TO...
CHIP
BOOTSTRAP! Bootstrap may take a few minutes; please be patient.
Downloading and installing packages into local source directory:
  Setting up CIPD package manager...done
  Setting up Python environment....done
  Setting up Host tools.....done
Activating environment (setting environment variables):
  Setting environment variables for CIPD package manager...done
  Setting environment variables for Python environment....done
  Setting environment variables for Host tools.....done
Sanity checking the environment:
20201125 12:57:40 INF Environment passes all checks!
Environment looks good, you are ready to go!
To activate this environment in the future, run this in your terminal:
source ./activate.sh
simran@simran-Ubuntu:~/connectedhomeip$
```

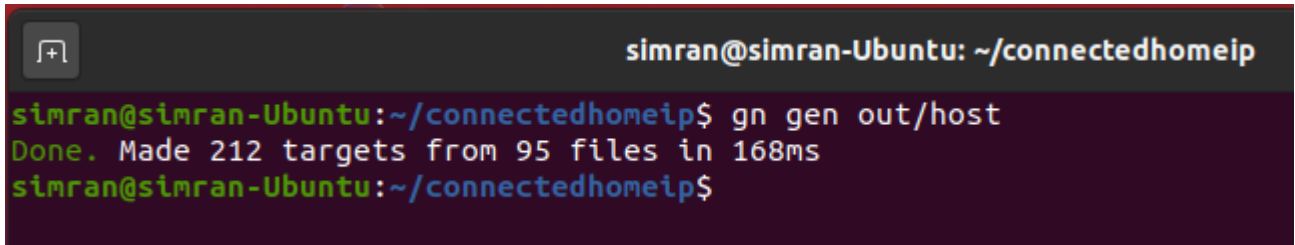
If this script says the environment is out of date, it can be updated by running:

```
source scripts/bootstrap.sh
```

This script re-creates the environment from scratch, which is expensive, so avoid running it unless the environment is out of date.

2. Make a build directory:

```
gn gen out/host
```

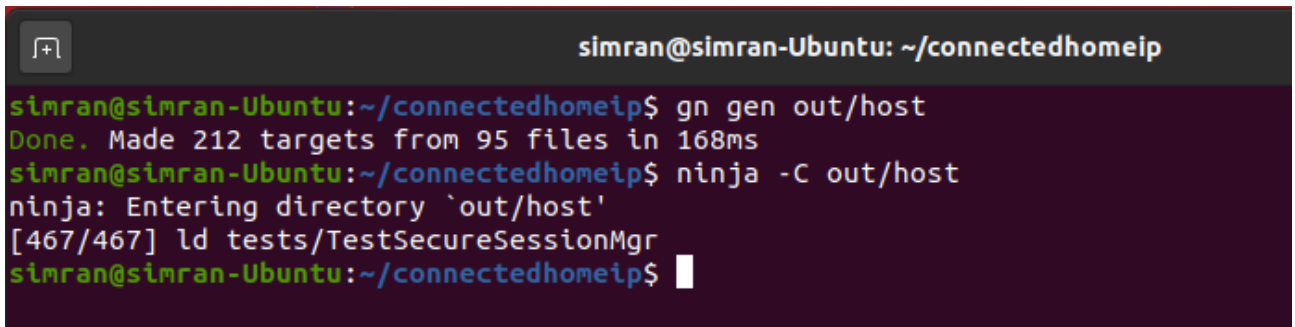


```
simran@simran-Ubuntu: ~/connectedhomeip
simran@simran-Ubuntu:~/connectedhomeip$ gn gen out/host
Done. Made 212 targets from 95 files in 168ms
simran@simran-Ubuntu:~/connectedhomeip$
```

3. To build these files, pass the label with its path (but no leading “/”) to ninja:\



```
ninja -C out/host
```



```
simran@simran-Ubuntu: ~/connectedhomeip
simran@simran-Ubuntu:~/connectedhomeip$ gn gen out/host
Done. Made 212 targets from 95 files in 168ms
simran@simran-Ubuntu:~/connectedhomeip$ ninja -C out/host
ninja: Entering directory `out/host'
[467/467] ld tests/TestSecureSessionMgr
simran@simran-Ubuntu:~/connectedhomeip$
```

The directory name ‘**out/host**’ can be any directory, although it's conventional to build within the out directory. This example uses host to emphasize that we're building for the host system. Different build directories can be used for different configurations, or a single directory can be used and reconfigured as necessary via gn args.

3 Creating EFR32 Lock-App

Note: Instructions provided in this **section 3** can be used to build Chip project on WSL2 (Windows 10), Native Linux or Mac machine)

The EFR32 lock example provides a baseline demonstration of a door lock device, built using CHIP and the Silicon Labs gecko SDK. The example currently supports Open Thread.

The lock example is intended to serve both to explore the workings of CHIP as well as a template for creating real products based on the Silicon Labs platform.

3.1 Building the example

1. Switch to efr32 lock-app folder.

```
cd ~/connectedhomeip/examples/lock-app/efr32
```

2. Verify all the submodules are updated and synchronized.

```
git submodule update --init
```

3. Make sure to activate the environment. This will load gn and ninja for building further instructions.

```
source third_party/connectedhomeip/scripts/activate.sh
```

4. Setup the Gecko SDK 2.7 path. Refer to section 1.2 for downloading the Gecko SDK 2.7

```
export EFR32_SDK_ROOT=<path-to-silabs-sdk-v2.7>
```

```
export EFR32_SDK_ROOT=~/.sdk_support
```

5. Set the board number used to build this example project.

```
export EFR32_BOARD=BRD4170A
```

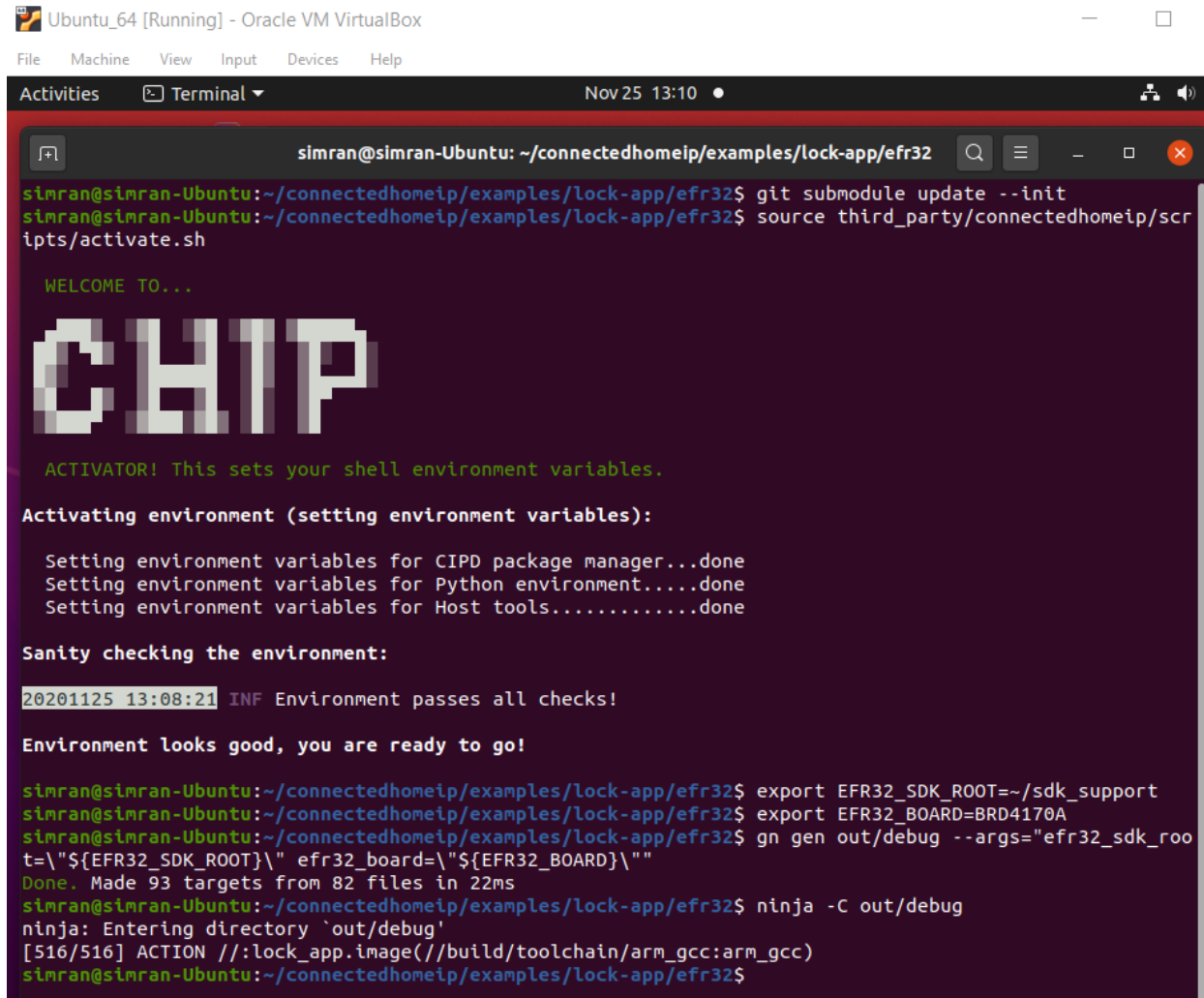
6. Now, setup the build directory before compiling the project. with GN, you can setup your own build directories with the settings you want. This lets you maintain as many different builds in parallel as you need.

```
gn gen out/debug --args="efr32_sdk_root=\"${EFR32_SDK_ROOT}\" efr32_board=\"${EFR32_BOARD}\""
```

7. To build the project, pass the label with its path to ninja

```
ninja -C out/debug
```

Creating EFR32 Lock-App



```

simran@simran-Ubuntu: ~/connectedhomeip/examples/lock-app/efr32
simran@simran-Ubuntu:~/connectedhomeip/examples/lock-app/efr32$ git submodule update --init
simran@simran-Ubuntu:~/connectedhomeip/examples/lock-app/efr32$ source third_party/connectedhomeip/scripts/activate.sh

WELCOME TO...

CHIP

ACTIVATOR! This sets your shell environment variables.

Activating environment (setting environment variables):

Setting environment variables for CIPD package manager...done
Setting environment variables for Python environment....done
Setting environment variables for Host tools.....done

Sanity checking the environment:

20201125 13:08:21 INF Environment passes all checks!

Environment looks good, you are ready to go!

simran@simran-Ubuntu:~/connectedhomeip/examples/lock-app/efr32$ export EFR32_SDK_ROOT=~/sdk_support
simran@simran-Ubuntu:~/connectedhomeip/examples/lock-app/efr32$ export EFR32_BOARD=BRD4170A
simran@simran-Ubuntu:~/connectedhomeip/examples/lock-app/efr32$ gn gen out/debug --args="efr32_sdk_root=\"$EFR32_SDK_ROOT\" efr32_board=\"$EFR32_BOARD\""
Done. Made 93 targets from 82 files in 22ms
simran@simran-Ubuntu:~/connectedhomeip/examples/lock-app/efr32$ ninja -C out/debug
ninja: Entering directory `out/debug'
[516/516] ACTION //:lock_app.image(//build/toolchain/arm_gcc:arm_gcc)
simran@simran-Ubuntu:~/connectedhomeip/examples/lock-app/efr32$

```

8. If you want to delete the generated the project files: *(not required for this lab)*

```
rm -rf out/
```

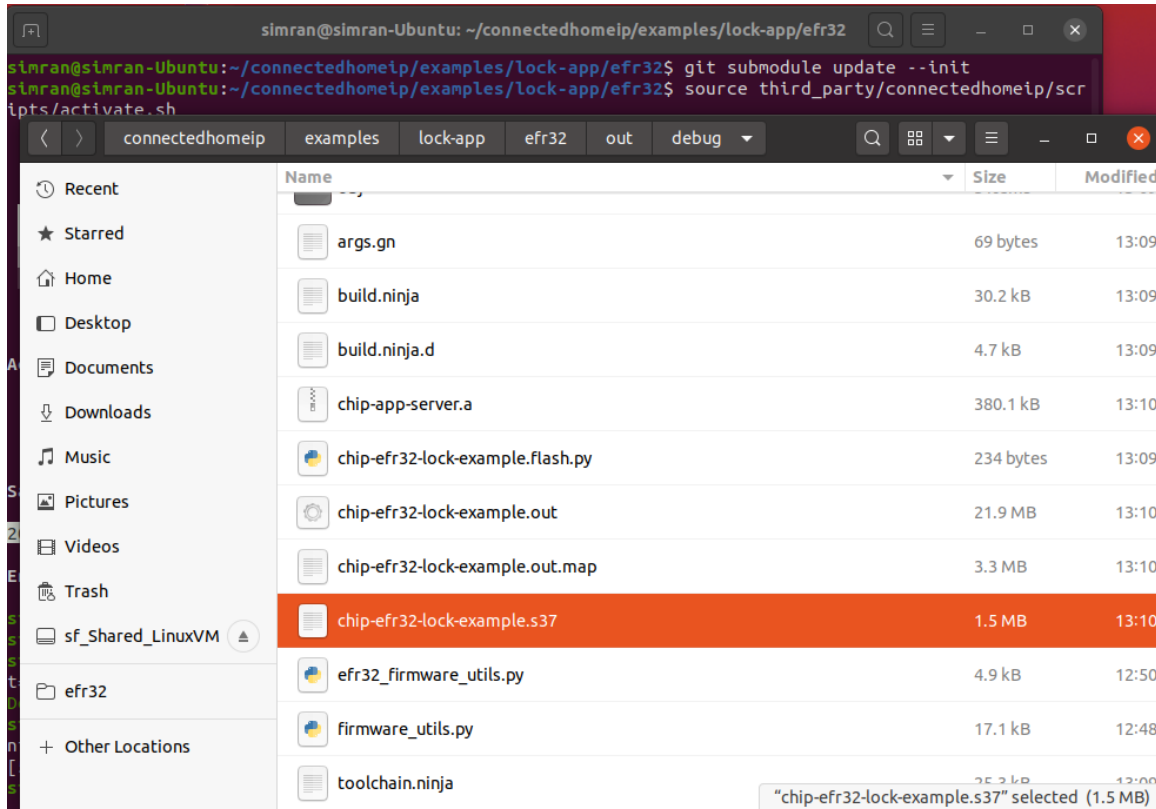
Creating EFR32 Lock-App

3.2 Flashing the Firmware

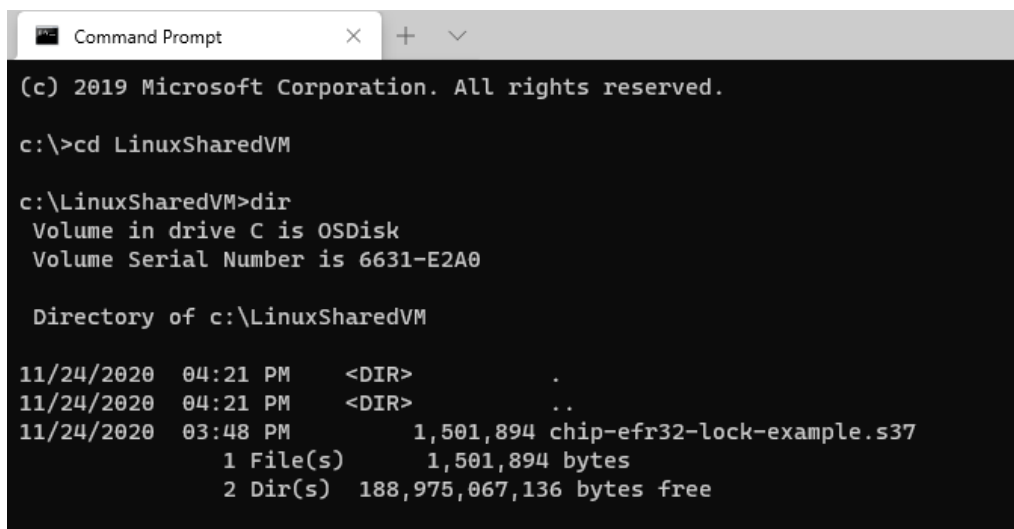
For this lab we will begin by programming the BRD4170A using the wired USB interface.

The generated files for 'Lock-app' project on Linux VM is typically be stored at:

`home\connectedhomeip\examples\lock-app\efr32\out\debug`



1. Copy the “**chip-efr32-lock-example.s37**” from .../out/debug folder from Linux VM to Windows 10 using shared folder.
2. Open Command prompt on Windows 10, make sure you are in directory where firmware file is copied in previous step.



Creating EFR32 Lock-App

- Before flashing the application firmware, verify whether the bootloader is flashed. Otherwise the application will not bootup.
- Type the following command to flash the firmware to the BRD4170A:

```
commander.exe flash chip-efr32-lock-example.s37
```

```

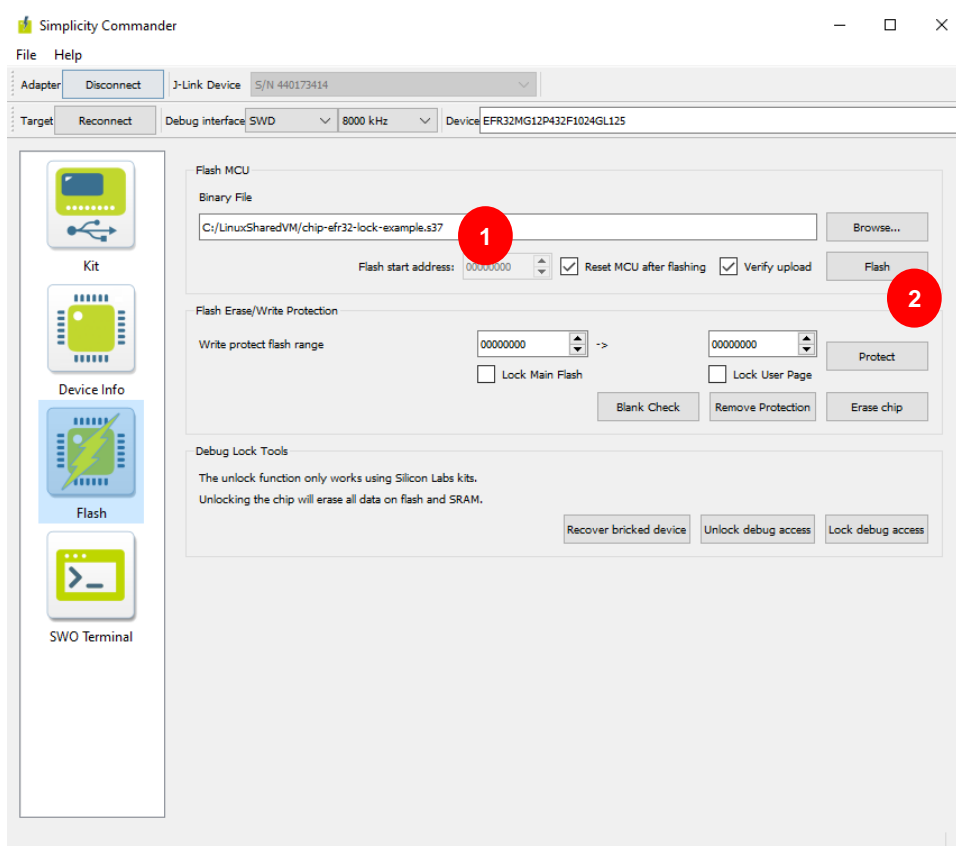
Microsoft Windows [Version 10.0.18362.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

c:\>cd LinuxSharedVM

c:\LinuxSharedVM>commander flash chip-efr32-lock-example.s37
Parsing file chip-efr32-lock-example.s37...
Writing 524288 bytes starting at address 0x00000000
Comparing range 0x00000000 - 0x0001FFFF (128 KB)
Comparing range 0x00020000 - 0x0003FFFF (128 KB)
Comparing range 0x00040000 - 0x0005FFFF (128 KB)
Comparing range 0x00060000 - 0x0007FFFF (128 KB)
Programming range 0x00000000 - 0x000007FF (2 KB)
Programming range 0x00000800 - 0x00000FFF (2 KB)
Programming range 0x00001000 - 0x000017FF (2 KB)
Programming range 0x00001800 - 0x00001FFF (2 KB)
Programming range 0x00070000 - 0x000707FF (2 KB)
Programming range 0x00070800 - 0x00070FFF (2 KB)
Verifying range 0x00000000 - 0x0001FFFF (128 KB)
Verifying range 0x00020000 - 0x0003FFFF (128 KB)
Verifying range 0x00040000 - 0x0005FFFF (128 KB)
Verifying range 0x00060000 - 0x0007FFFF (128 KB)
DONE

c:\LinuxSharedVM>
  
```

Note: You can also manually launch GUI interface of commander.exe file from (C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander) and flash the file to the board

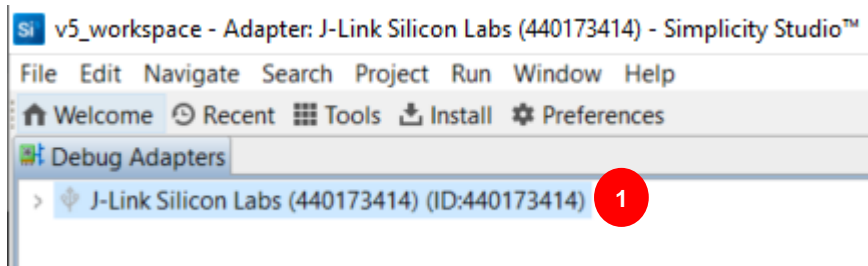


4 Preparing OTBR and RCP Device

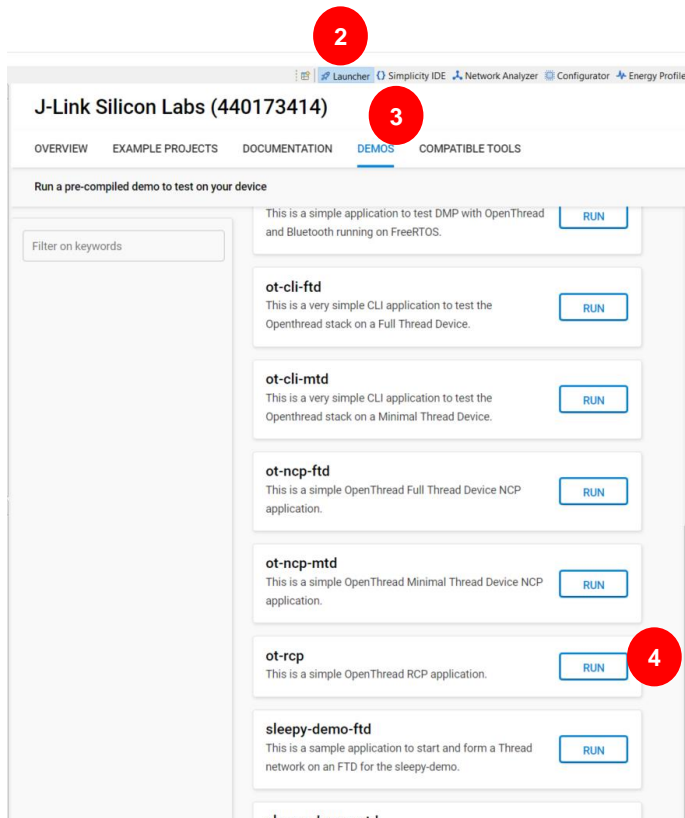
This section is not covered in this tutorial. So, we will only cover brief setup of RCP and Board Router. For in-depth detail on setting up Open Thread Border Router, you can refer to *AN1256: Using the Silicon Labs RCP with the Open Thread Border Router* <https://www.silabs.com/documents/public/application-notes/an1256-using-sl-rcp-with-openthread-border-router.pdf>

4.1 Preparing RCP Device

1. Once you have your WSTK kit with BRD4170A board connected via USB, Simplicity Studio 5 will detect the available adapters. Select the adapter by clicking over it.



2. Click on the 'Launcher' tab on Simplicity Studio 5



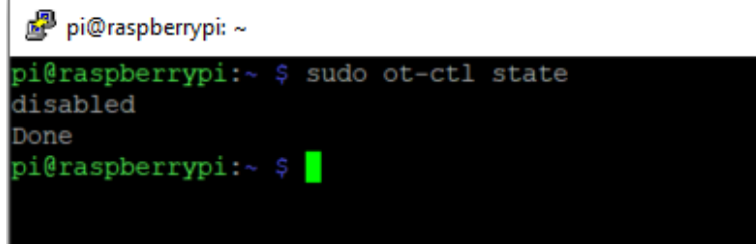
3. Click on 'Demo' tab to list all available pre-compiled firmware for selected device.
4. Click 'RUN' on 'ot-rpc' demo project. This will install the pre-built RCP image for selected board.
5. Now RCP device is ready to connect with OTBR running on Raspberry Pi.

4.2 Launching OTBR (Open Thread Boarder Router)

Assuming OTBR is already setup and running on Raspberry pi (refer to section 4 introduction for installing OTBR on Raspberry Pi). You can use SSH terminal like Putty. Open SSH terminal and login to Raspberry pi.

4.2.1 Check the status of OTBR by typing following command

```
sudo ot-ctl state
```

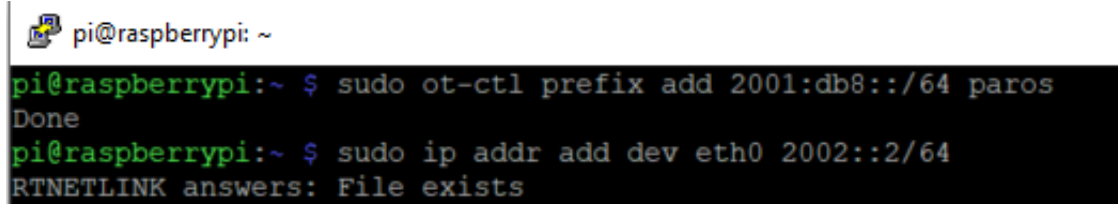


```
pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo ot-ctl state
disabled
Done
pi@raspberrypi:~ $
```

4.2.2 OT-BR need to have a Global prefix configured. Type the following commands on RPI

```
sudo ot-ctl prefix add 2001:db8::/64 paros
```

```
sudo ip addr add dev <interface> 2002::2/64 // e.g. name of interface that Raspi
// is connected to like eth0/wpan0/etc
```



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo ot-ctl prefix add 2001:db8::/64 paros
Done
pi@raspberrypi:~ $ sudo ip addr add dev eth0 2002::2/64
RTNETLINK answers: File exists
```

4.2.3 Restart the OTBR, type the following command

```
sudo ot-ctl state
```

```
> thread stop
```

```
> ifconfig down
```

```
> ifconfig up
```

```
> thread start
```

Preparing OTBR and RCP Device

```

pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo ot-ctl
> thread stop
Done
> ifconfig down
Done
> ifconfig up
Done
> thread start
Done
> state
leader
Done
>

```

4.2.4 Check the IP address of OTBR on Raspberry Pi to verify the changes.

```
> ipaddr
```

```

pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo ot-ctl
> ipaddr
2001:db8:0:0:f74e:8538:d8da:b15a
fdde:ad00:beef:0:0:ff:fe00:fc00
fdde:ad00:beef:0:0:ff:fe00:7c00
fdde:ad00:beef:0:0:6180:6a0e:4092:8538
fe80:0:0:0:f3:3919:a497:7a99
Done
>

```

4.2.5 Get the connection detail of OTBR. This is required for connecting child device with OTBR.

Type following commands:

```

> channel
> panid
> masterkey

```

```

pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo ot-ctl
> channel
11
Done
> panid
0xface
Done
> masterkey
00112233445566778899aabbccddeeff
Done
>

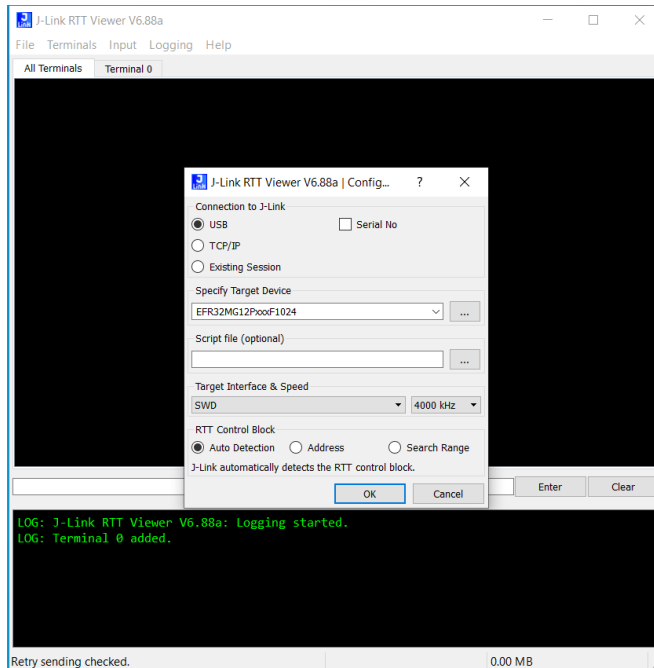
```

Take a note of these details and it will be used in [section 5.2.3](#) for connecting CHIP End-device.

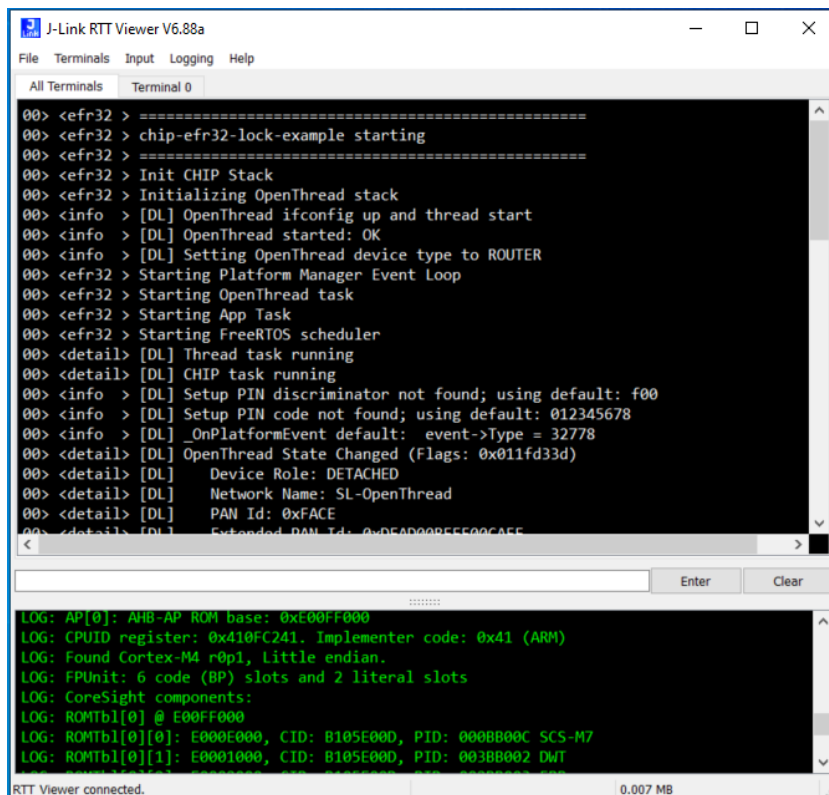
5 Running the CHIP End Device Demo

5.1 Startup the J-Link RTT Viewer

1. On Windows 10, connect CHIP End-device to J-Link RTT viewer, Press OK to connect.



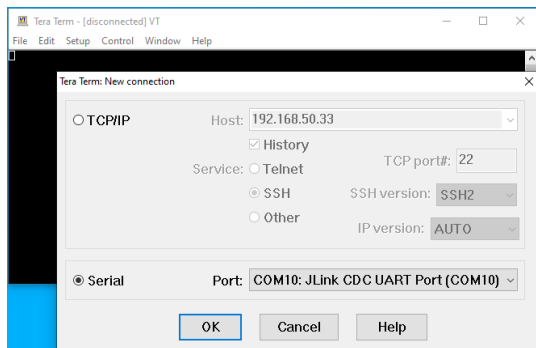
2. Press the reset button on WSTK kit and logs will be shown on J-Link RTT viewer when Chip End device restarts.



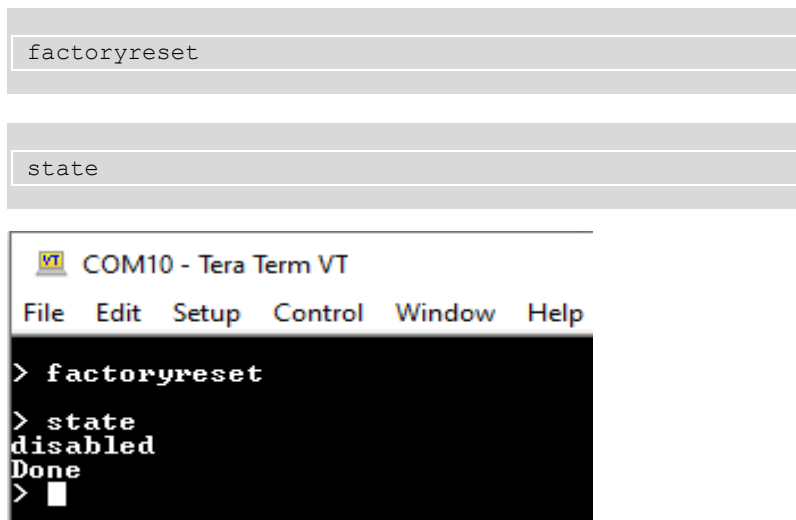
Running the CHIP End Device Demo

5.2 Open Tera Term serial terminal for CHIP End-device

5.2.1 On Windows 10, Open Serial Terminal like Tera term and connect the end-device.

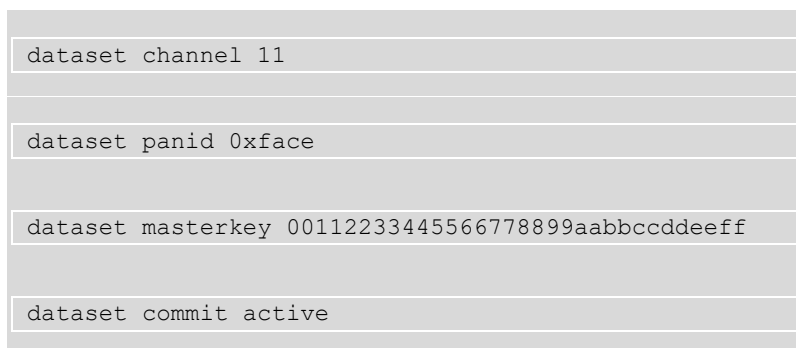


5.2.2 Verify the state of the device, type 'factoryreset' followed by 'state' on Tera-term console



5.2.3 Connect the CHIP end device to OTBR, connection detail can be obtained as explained in [Section 4.2.5](#).

Type following commands:



```

> dataset channel 11
Done
> dataset panid 0xfac
Done
> dataset masterkey 00112233445566778899aabbccddeeff
Done
> dataset commit active
Done
> ifconfig up
Done
> thread start
Done

```

5.2.4 Check the IP address of CHIP end-device, it will have IPV6 address with “2001::db8” prefix

```

COM10 - Tera Term VT
File Edit Setup Control Window Help

> ipaddr
fdde:ad00:beef:0:0:ff:fe00:2400
2001:db8:0:0:cf3d:172e:c2fb:ee2e
fe80:0:0:0:484d:e74f:47ef:1000
fdde:ad00:beef:0:be65:37a6:3abf:b7cc
Done
> 

```

5.2.5 On raspberry pi, verify if OTBR have child device

```

pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo ot-ctl
> child table
| ID | RLOC16 | Timeout | Age | LQ In | C_VN | R|S|D|N| Extended MAC
+---+-----+-----+---+-----+-----+-----+-----+-----+
| 2 | 0x7c02 | 240 | 29 | 3 | 206 | 1|1|1|1| 4a4de74f47ef0f
00 |
Done

```

5.2.6 To verify, whether the device communication is successful send ping from raspberry pi to child device.

```

pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo ot-ctl
> ping 2001:db8:0:0:cf3d:172e:c2fb:ee2e
Done
> 16 bytes from 2001:db8:0:0:cf3d:172e:c2fb:ee2e: icmp_seq=2 hlim=255 time=35ms

```

Now, CHIP end-device is successfully on thread network.

5.3 Building CHIP-TOOL in Linux VM

Note: Instruction provided in this **section 5.3** can be used to build Chip-tool on WSL2 (Windows 10), Native Linux or Mac machine) But WSL2 cannot act as client to send messages using chip-tool

Chip-tool allow to send message over thread network to control CHIP END-DEVICE. Before using chip-tool, we must build binaries just like lock-app example.

On Virtual Machine, open Linux terminal, and switch to chip-tool directory under '**connectedhomeip**' directory.

```
cd examples/chip-tool
```

```
git submodule update --init
```

```
source third_party/connectedhomeip/scripts/activate.sh
```

```
gn gen out/debug
```

```
ninja -C out/debug
```

Now, generated binaries for chip-tool are under '**~/examples/chip-tool/out/debug**' folder. Using chip-tool you can now control the lock status with on/off command. But before that we need to add ipv6 route in next step.

5.4 ADD IPV6 Route on Client Device

Client device is used to send messages to Chip End device. Linux VM or native Linux or Mac machine can be used as client device.

Note: WSL2 does not currently support IPV6 routing and cannot be act as client for sending messages using chip-tool.

- Type the following command on VM (Ubuntu) (or native Linux Ubuntu machine)

```
sudo ifconfig <interface> inet6 add 2002::1/64
```

```
sudo ip route add 2001:db8:0:0::/64 via 2002::2
```

```
sudo ip route add fdde:ad00:beef::/64 via 2002::2
```

Running the CHIP End Device Demo

```

simran@simran-Ubuntu: ~/connectedhomeip/examples/chip-tool
simran@simran-Ubuntu:~/connectedhomeip/examples/chip-tool$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.50.22 netmask 255.255.255.0 broadcast 192.168.50.255
    inet6 fe80::479a:1f70:a993:847b prefixlen 64 scopeid 0x20<link>
    ether 08:00:27:36:58:9a txqueuelen 1000 (Ethernet)
    RX packets 3280 bytes 3766406 (3.7 MB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 2900 bytes 288157 (288.1 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
    inet6 ::1 prefixlen 128 scopeid 0x10<host>
    loop txqueuelen 1000 (Local Loopback)
    RX packets 236 bytes 21567 (21.5 KB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 236 bytes 21567 (21.5 KB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

simran@simran-Ubuntu:~/connectedhomeip/examples/chip-tool$ sudo ifconfig enp0s3 inet6 add 2002::1/64
simran@simran-Ubuntu:~/connectedhomeip/examples/chip-tool$ sudo ip route add 2001:db8:0:0::/64 via 2002::2
simran@simran-Ubuntu:~/connectedhomeip/examples/chip-tool$ sudo ip route add fdde:ad00:beef::/64 via 2002::2
simran@simran-Ubuntu:~/connectedhomeip/examples/chip-tool$

```

- On Mac machine, following commands can be used to add ipv6 routes

```
sudo ifconfig <interface> inet6 add 2002::1/64 //e.g enp0s3
```

```
sudo route add -inet6 2001:db8:0:0::/64 2002::2
```

```
sudo route add -inet6 fdde:ad00:beef::/64 2002::2
```

To verify the route, ping the ipv6 address of CHIP-end device from Linux VM

```

simran@simran-Ubuntu: ~
simran@simran-Ubuntu:~$ ping 2001:db8:0:0:cf3d:172e:c2fb:ee2e
PING 2001:db8:0:0:cf3d:172e:c2fb:ee2e(2001:db8::cf3d:172e:c2fb:ee2e) 56 data byt
es
64 bytes from 2001:db8::cf3d:172e:c2fb:ee2e: icmp_seq=1 ttl=254 time=103 ms
64 bytes from 2001:db8::cf3d:172e:c2fb:ee2e: icmp_seq=2 ttl=254 time=56.8 ms
^C
--- 2001:db8:0:0:cf3d:172e:c2fb:ee2e ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1016ms
rtt min/avg/max/mdev = 56.835/79.995/103.156/23.160 ms
simran@simran-Ubuntu:~$

```

Note: Chip-tool is currently working on Linux VM/ native Linux machine or Mac machine. WSL2 on Windows 10 does not fully support bridge network mode to run chip-tool via ipv6 route.

5.5 Send message using Chip-tool to End-Device

- Currently, WSL2 cannot work as client to send messages using chip-tool.

To use the Client to send a CHIP commands, run the built executable and pass it the target cluster name, the target command name, the IP address and port of the server to talk to as well as an endpoint id. The endpoint id must be between 1 and 240.

Running the CHIP End Device Demo

1. On VM, open Linux terminal window, make sure we are in the '`~/connectedhomeip/examples/chip-tool/out/debug`' directory.
2. Using chip-tool, control the lock status with on/off command.

```
./chip-tool onoff on <ipv6 address of the node> 11097 1
```

```

simran@simran-Ubuntu: ~/connectedhomeip/examples/chip-tool/out/debug
CHIP:T00: status: EMBER_ZCL_STATUS_SUCCESS (0x00)
CHIP:CTL: Shutting down the controller
simran@simran-Ubuntu:~/connectedhomeip/examples/chip-tool/out/debug$ ./chip-tool onoff on 2001:db8:0:0:cf3d:172e:c2fb:ee2e 1
1097 1
CHIP:T00: OnConnect
CHIP:T00: Endpoint id: '0x01', Cluster id: '0x0006', Command id: '0x01'
CHIP:ZCL: Successfully encoded 13 bytes
CHIP:T00: Encoded data of length 16
CHIP:IN: New pairing for key 0!!
CHIP:IN: Secure transport transmitting msg 0 after encryption
CHIP:CTL: SendMessage returned 0
CHIP:DL: CHIP task running
CHIP:DL: wpa supplicant: connected to wpa supplicant proxy
CHIP:T00: OnMessage: Received 18 bytes
CHIP:T00: APS frame processing success!
CHIP:T00: DefaultResponse (0x0B):
CHIP:T00:   commandId: 0x01
CHIP:T00: status: EMBER_ZCL_STATUS_SUCCESS (0x00)
CHIP:CTL: Shutting down the controller
simran@simran-Ubuntu:~/connectedhomeip/examples/chip-tool/out/debug$

```

On J-Link RTT Viewer connected on CHIP End-Device, you can verify the command is successfully executed. Also, on WSTK you can see Led is turned ON.

```

J-Link RTT Viewer V6.88a
File Terminals Input Logging Help
All Terminals Terminal 0
00> <detail> [DL] src 2002::1, port 11097
00> <detail> [DL] dest 2001:DB8::CF3D:172E:C2FB:EE2E, port 11097
00> <info> [SVR] Packet received from UDP:2002::1:11097: zu bytes
00> <info> [ZCL] APS frame processing success!
00> <info> [ZCL] RX len 3, ep 1, clus 0x 6
00> <info> [ZCL] FC 1 seq 1 cmd 1 payload[
00> <info> [ZCL] NULL
00> <info> [ZCL] ]
00> <info> [ZCL] On/Off set value: 1 1
00> <info> [ZCL] Toggle on/off from 0 to 1
00> <efr32> Lock Action has been initiated
00> <detail> [ZCL] Measured APS frame size 13
00> <detail> [ZCL] Successfully encoded 13 bytes
00> <detail> [IN] Secure transport transmitting msg 2 after encryption
00> <detail> [DL] Thread packet SENT: UDP, len 114
00> <detail> [DL] src 2001:DB8::CF3D:172E:C2FB:EE2E, port 11097
00> <detail> [DL] dest 2002::1, port 11097
00> <info> [ZCL] T 0:TX (0x77630) Ucast 0x00x7bc38
00> <info> [ZCL] TX buffer: [
00> <info> [ZCL] 0hX0hX0hX0hX0hX
00> <info> [ZCL] ]
00> <info> [ZCL] Data model processing success!
00> <efr32> Lock Action has been completed

```

This complete the tutorial on building CHIP Lock-app. End device can establish communication with OTBR and able to send messages to CHIP end device via client running on Linux VM.