PATHPARTNER

DecaWave: UWB-IOT tools Infrastructure
OT-DW1000 UserGuide

Table of Contents

1. About this Manual

This document describes the steps to build and work on OpenThread with communication-centric UWB radio MAC layer on EVB1000 Radio and Nordic NRF52840 Hardware platform. OpenThread with UWB MAC package serves to provide a software platform for development, deployment and execution of OpenThread CLI, NCP application.

In this context, the document contains instructions to:

Setting up the Environment

Build the OpenThread CLI and NCP applications

Flash the OpenThread CLI and NCP applications

2. Getting Started

2.1. System Requirements

Refer to 'Dependencies' section of release notes

2.2. Running OT CLI and NCP Application on EVB1000 Radio and NRF52840 platform

This section describes how to run the OpenThread CLI and NCP application with UWB MAC

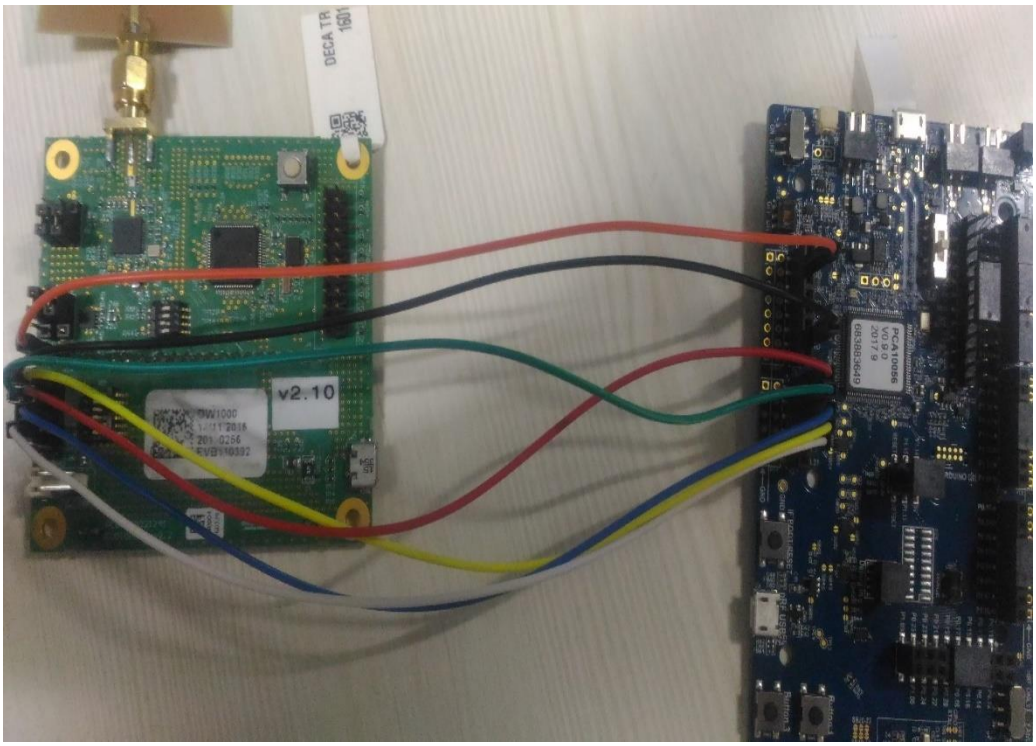
2.2.1. Common Steps for all examples

- Option 1
  - To initialize the Environment Variables, installing Dependent Libraries (First time), setup (clone the OT repo and sym link creation for dw1000) and building the application
    - Inside the cloned repo,run the script – "**./build_setup.sh INITIAL**"
  - To initialize the Environment Variables, setup (clone the OT repo and sym link creation for dw1000) and building the application (Next Time on Wards)
    - Inside the cloned repo,run the script – "**./build_setup.sh Update**"
  - To build the OT CLI and NCP application
    - Inside the cloned repo run the script – "**./build_setup.sh**"
- Option 2
  - Initialize the Environment Variables
    - Inside the cloned repo,run the script – "***source setenv.sh***"
  - Installing Dependent Libraries (First time)
    - Inside the cloned repo run the script - "**./setup.sh INITIAL**"
  - Setup (clone the OT repo and sym link creation for dw1000) without installing Dependent Libraries (Next Time on Wards)
    - Inside the cloned repo run the script – "**./setup.sh UPDATE**"
  - Build the OT CLI and NCP application
    - Inside the cloned repo run the script – "./build_setup.sh"
- Installing JLink Flasher/Debugger
  - Download the package from https://www.segger.com/downloads/jlink (for ubuntu,download debian 64bit)
  - Install the package either by double clicking on package or through command line Interface as "dpkg -i package_name"
- To connect EVB1000 board to nRF52840 PDK
- Turn off S2 DIP switches
- Turn off S3-1 DIP switch
- Remove the jumper J10
- Make sure the pins of nRF52840 PDK and EVB1000 Device are blue wired as defined below

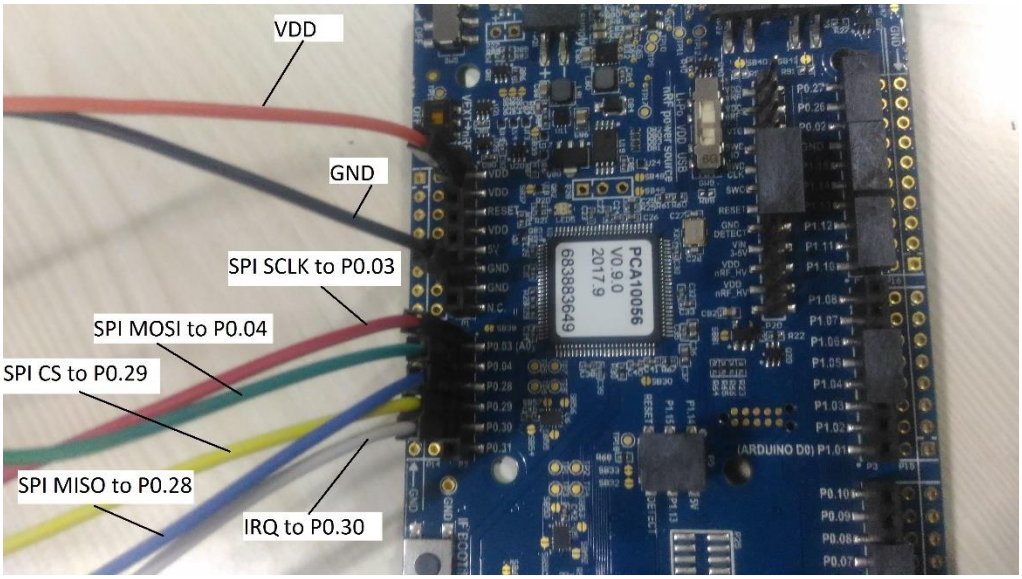
EVB 1000 to Nordic NRF52840

PIN	EVB1000	NORDIC NRF52840
IRQ	J6: Pin4	P0.30
MISO	J6: Pin5	P0.28
MOSI	J6: Pin8	P0.4
SCLK	J6: Pin7	P0.3
CS	J6: Pin9	P0.29
GND	J6: Pin10	GND
VDD	J10: Pin2	VDD

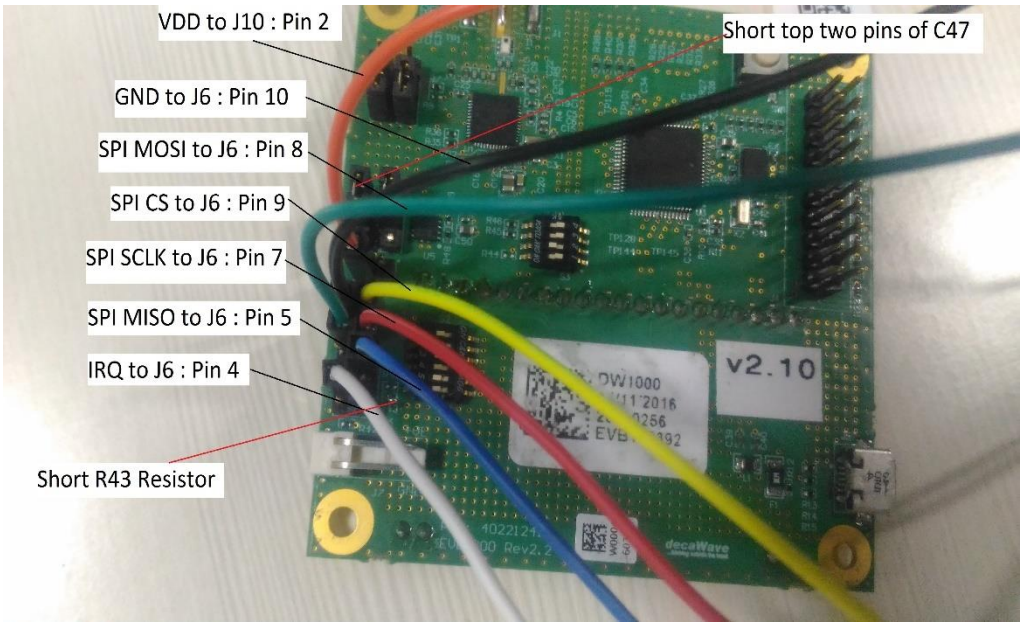
2.2.2. Connecting Nordic Hardware to EVM with SPI Interface



Connectors for SPI Interface on NRF52840 Nordic Platform to EVB1000



Connection Details of SPI Interface on NRF52840 Nordic Platform



Connection Details of SPI Interface on EVB1000

2.2.3. To Flash the OpenThread CLI and NCP Application

- "cd nrfjprog"
- "sudo ./nrfjprog -f nrf52 --chiperase --program <path-to-the-binary>"
  - For example: To Flash OT CLI Binary -
  - "sudo ./nrfjprog -f nrf52 -r --chiperase --program ../output/bin/arm-none-eabi-ot-cli-ftd.hex"
  - To reset the Board – "sudo ./nrfjprog -f nrf52 -r"

2.2.4. Running OpenThread CLI Application

- Flash the CLI binary (arm-none-eabi-ot-cli-ftd.hex) on nrf52840 board as defined in section 2.2.2
- open a new terminal (say "Terminal-1") and input
- "cd tools/pyterm"

- "sudo ./pyterm -p /dev/ttyACM0"
- open a new terminal (say "Terminal-2") and input
- "cd tools/pyterm"
- "sudo ./pyterm -p /dev/ttyACM1"

Note:

In Linux, if we connect three nodes using USB Port, the nodes will be detected as /dev/ttyACM0(say "Node 1") or /dev/ttyACM1(say "Node 2") or /dev/ttyACM2 (say "Node 3")

- To Start a new Thread network on terminal1 which is connected to NRF52840

```
> panid 0xdeca
  Done
> ifconfig up
  Done
> thread start
  Done
```

- Wait for few seconds(say "10secs") then the node will become a Leader of the network.

```
> state
 Leader
```

- Open a terminal2 which is connected to other NRF52840 board and then attach a node to the network.

```
> panid 0xdeca
    Done
> ifconfig up
    Done
> thread start
    Done
```

- Wait for few seconds (say "10Secs") then the second node will be attached and shall function as a child node.

```
> state
   child
```

- List all IPv6 addresses of the first node which is a Leader

```
> ipaddr
 fdde:ad00:beef:0:0:ff:fe00:fc00
 fdde:ad00:beef:0:0:ff:fe00:9c00
 fdde:ad00:beef:0:4bcb:73a5:7c28:318e
 fe80:0:0:0:5c91:c61:b67c:271c
```

Note:

- IPv6 address with suffix "**fd**" is mesh-local address.
- A Mesh-local address types containing "**ff:fe00**" is Router Locator (RLOC)
- A Mesh-local address types containing "**ff:fe00**" is Endpoint Identifier (EID)
- Identify the EID in the list of ipaddr.
- For Example, EID - " fdde:ad00:beef:0:4bcb:73a5:7c28:318e"
- IPv6 address with suffix "fe80" is link-local address
- Choose one of the node(Either Leader/Child) and send an ICMPv6 ping from the other Node (Say from Leader node)

```
> ping fdde:ad00:beef:0:0:ff:fe00:fc00
  16 bytes from fdde:ad00:beef:0:0:ff:fe00:fc00: icmp_seq=1 hlim=64
time=8ms
```

- To Ping repeatedly, use the command as
  " ping ipv6_address <payloadlength> <count> <timegap>"
- Example: ping fdde:ad00:beef:0:0:ff:fe00:fc00 8 10 2

For More details on How to use Open Thread CLI Application for different commands Please refer this links: OpenThread CLI Reference README.md

2.2.5. Running OpenThread NCP Application

2.2.5.1. Introduction

NCP is a low-power Wireless Network Co-Processor, a Border Router to connect the thread network with external network using ***wpantund*** Interface running on the host (Linux PC). ***wpantund*** is a user-space network interface driver/daemon that provides a native IPv6 network interface to a low-power wireless Network Co-Processor (or NCP).

2.2.5.2. Running NCP with wpantund

- Flash the NCP binary (arm-none-eabi-ot-ncp.hex) on nrf52840 board as defined in section 2.2.2.
- Open a Terminal and input "sudo /usr/local/sbin/wpantund -o NCPSocketName /dev/ttyACM0 -o WPANInterfaceName wpan0"
- This command will run the Wpantund Interface on the Linux PC with terminal as wpan0 and initializes the NCP.
- Open a new Terminal and input "sudo wpanctl -I wpan0"
- This command will start the wpan control interface (wpanctl:wpan0>) to configure and control NCP node.
- To check the status of the NCP input "status". Initial output is expected as below

```
wpan0 => [
    "NCP:State" => "offline"
    "Daemon:Enabled" => true
    "NCP:Version" => "OPENTHREAD/0.01.00;NRF52840; Mar 27 2017 10:06:23"
"Daemon:Version" => "0.08.00 (/f911961-dirty; Mar 31 2017 11:32:13)"
"Config:NCP:DriverName" => "spinel"
"NCP:HardwareAddress" => [B8DD443A6E973DEB]
```

2.2.5.2.1. To configure NCP as router and CLI node as Leader

- Start a CLI node as specified in the section 2.2.3 in a separate terminal (panid 0xdeca, ifconfig up , thread start), so it will become Leader of the newly formed network.
- To Join NCP Node to the above network, use "scan" and "Join" commands.
  wpanctl:wpan0> scan

  Output will be:

```
| Joinable | NetworkName | PAN ID | Ch | XPanID | HWAddr | RSSI
---+----------+------------------+--------+----+-----------------+-------------  ----
+------
1 | NO | "OpenThread" | 0xDECA | 5 | DEAD00BEEF00CAFE |
AA9D0AFEC741A253 | -67

wpanctl:wpan0> join 1 -T 2
    Joining "OpenThread" DEAD00BEEF00CAFE as node type "router"
   Successfully Joined!
```

- To check the status, wpanctl:wpan0> status
  wpan0 => [
  "NCP:State" => "associated"
  "Daemon:Enabled" => true
  NCP:Version" => "OPENTHREAD/0.01.00; none; May 12 2017 11:47:38"
  "Daemon:Version" => "0.08.00d (/f911961-dirty; May 5 2017 10:52:26)"
  "Config:NCP:DriverName" => "spinel"
  "NCP:HardwareAddress" => [18B4300000000002]
  "NCP:Channel" => 5
  "Network:NodeType" => "router"
  "Network:Name" => "Open Thread"
  "Network:XPANID" => 0x64941B1C5D9C8F10
  "Network:PANID" => 0xDECA
  "IPv6:LinkLocalAddress" => "fe80::24b1:b3df:dc48:9585"
  "IPv6:MeshLocalAddress" => "fd64:941b:1c5d:0:7c34:ca08:8028:352c"
  "IPv6:MeshLocalPrefix" => "fd64:941b:1c5d::/64"
  "com.nestlabs.internal:Network:AllowingJoin" => false
  ]

- In cli terminal input "ping fd64:941b:1c5d:0:7c34:ca08:8028:352c"
  16 bytes from fd64:941b:1c5d:0:7c34:ca08:8028:352c icmp_seq=1 hlim=64 time=121ms

- Open a new Linux Terminal and input
  "ping6 fd64:941b:1c5d:0:7c34:ca08:8028:352c"

  PING
  fd64:941b:1c5d:0:7c34:ca08:8028:352c(fd64:941b:1c5d:0:7c34:ca08:8028:352c) 56 data bytes
  64 bytes from fd64:941b:1c5d:0:7c34:ca08:8028:352c: icmp_seq=1 ttl=64 time=0.015 ms
  64 bytes from fd64:941b:1c5d:0:7c34:ca08:8028:352c: icmp_seq=1 ttl=64 time=0.015 ms
  64 bytes from fd64:941b:1c5d:0:7c34:ca08:8028:352c: icmp_seq=1 ttl=64 time=0.015 ms

2.2.5.2.2. Making NCP as Leader and CLI node as end-device/Child

- Start NCP node as mentioned in Sec 2.2.5.3
- Input "form" to make NCP form a Network and become Leader
  wpanctl:wpan0> form Thread_Name

- Forming WPAN "Thread_Name" as node type "router"
  Successfully formed!

- Check the status.
  wpanctl:wpan0> status
  wpan0 => [
  "NCP:State" => "associated"
  "Daemon:Enabled" => true
  "NCP:Version" => "OPENTHREAD/0.01.00; none; May 12 2017 11:47:38"
  "Daemon:Version" => "0.08.00d (/f911961-dirty; May 5 2017 10:52:26)"
  "Config:NCP:DriverName" => "spinel"
  "NCP:HardwareAddress" => [18B4300000000002]
  "NCP:Channel" => 19
  "Network:NodeType" => "leader"
  "Network:Name" => "Thread_Name"

"Network:XPANID" => 0x64941B1C5D9C8F10
"Network:PANID" => 0x803B
"IPv6:LinkLocalAddress" => "fe80::24b1:b3df:dc48:9585"
"IPv6:MeshLocalAddress" => "fd64:941b:1c5d:0:7c34:ca08:8028:352c"
"IPv6:MeshLocalPrefix" => "fd64:941b:1c5d::/64"
"com.nestlabs.internal:Network:AllowingJoin" => false
]

"getprop" will list all the parameters of the NCP Node
wpanctl:wpan0>getprop

- Now Run a CLI node with the same parameters observed on the NCP Assign Panid , channel and Masterkey
- CLI node will join the Network as a child. It can be ensured by using "state" command and "parent" commands on the CLI Node.
- On CLI Node input "Ping fd64:941b:1c5d:0:7c34:ca08:8028:352c"
  > 16 bytes from fd64:941b:1c5d:0:7c34:ca08:8028:352c icmp_seq=1 hlim=64 time=121ms

2.2.6. Running Border Router with Raspberry Pi with OT NCP and CLI Application

2.2.6.1. Introduction

Nordic semiconductors Developed an Example (Thread Border Router), i.e., Raspberry Pi as a Border Router connected with NCP node and to the Internet (Ethernet or Wi-Fi). Thread Border Router serves as a gateway between the Internet and the Thread network.

2.2.6.2. Test SetUp

- Raspberry Pi 3 B
- nRF52840 Development Kit
- 1 GB (or larger) microSD card with SD card adapter
- microUSB power supply for Raspberry Pi 3 B
- microUSB to USB cable for connecting the nRF52840 Development Kit to the Raspberry Pi
- Computer running the Linux operating system with SD card slot or SD/microSD USB adapter.

2.2.6.3. Dependencies Tools:

- Terminal client with serial connection support, such as PuTTy/Tera term/ Minicom /Pyterm
- Border Router Raspberry Pi image available from nRF5 SDK for Thread web page "nRF5 SDK for Thread v0.9.0-1.alpha web page" - RaspPi_Thread_Border_Router_Demo_v0.9.0-1.alpha.img
- NCP firmware binary – arm-none-eabi-ot-ncp.hex
- CLI firmware binary – arm-none-eabi-ot-cli-ftd.hex

2.2.6.4. Setup procedure:

- Complete the following steps to set up the Border Router solution:
- Flash the Raspberry Pi image to the microSD card
- Download the Raspbian image for thread from the direct link : https://www.nordicsemi.com/eng/nordic/download_resource/60502/12/57596032/118100

- Extract the package for Image file.
- Insert the SD Card into the SD card adapter and plug the adapter into ubuntu PC.
- Go to the command prompt and check the name given to sd card by using command
  sudo fdisk -l
- it may be /dev/sdb or /dev/sdc or /dev/sdd (varies from PC to PC)
- Use the following command to copy the image on to the Raspberry Pi.
  sudo dd if=<PATH to image>/2017-04-10-raspbian-jessie.img of=/dev/sdc bs=4M
- For Example:
  sudo dd if=~/Downloads/2017-04-10-raspbian-jessie.img of=/dev/sdc bs=4M
  sudo sync
- Eject the SD card safely.
- Insert the SD card into Raspberry Pi.
- Flash the NCP binary on to the Nordic Board connected with EVB1000 as defined in section 2.2.5

2.2.6.5. Starting the Border Router:

- Connect the NCP Node to the Raspberry Pi using micro USB CABLE as shown in the below set up figure.
- Connect the Raspberry Pi through an Ethernet cable to your switch/router that provides IPv4 or IPv6 connectivity with the DHCP or DHCPv6 service respectively. Connect the microUSB power supply to the Raspberry Pi. The Border Router will start.

Setup Details of NCP Node with Raspberry pi Platform

2.2.6.6. Thread Settings:

- The following are the default settings of the Border Router:
  Radio Channel: 11
  PAN ID: 0xABCD
  Network Master Key: 0x00112233445566778899AABBCCDDEEFF
  Mesh-Local Prefix: FDDE:AD00:BEEF::/64
  NAT64 Prefix: FD00:64:123:4567::/96
- To customize the Thread settings, you must Log into the Border Router using SSH.

- To Log into Raspberry Pi using SSH, requires its Ipv4 address.
- On your ububtu it is possible to find the IP address use the "nmap" command.
- Install the "nmap" command using
  sudo apt-get install nmap
- This method also requires your LAN ipv4 prefix. Use "ifconfig" command to find the prefix.
  ifconfig
  - For Example: if the inet address is 192.168.1.165 then 192.168.1.0 is your network prefix
- Now use this command, which lists all the IP addresses Present in your Network.
  > sudo nmap -sP 192.168.1.0/24
  Nmap scan report for 192.168.1.249
  Host is up (-0.100s latency).
  MAC Address: 00:25:64:B7:0F:01 (Raspberry Pi foundation)
- From the above output , Note the ipv4 address with the name Raspberry Pi foundation.
- Use the below command with username "root" and empty password to Log into Pi.
  ssh root@192.168.1.249
- Inside the Raspbian , vim/nano editors are available.
- To change the settings, edit the /etc/config/thread_border_router file. The following lines can be found at the top of the file.
  vi /etc/config/thread_border_router
  # NCP Configuration
  ncp_interface="wpan0"
  ncp_baudrate="115200"
  # Thread Network Configuration
  thread_network_key="00112233445566778899AABBCCDDEEFF"
  thread_channel="11"
  thread_panid="0xABCD"
  # IPv6 Addresses Configuration for NAT64
  prefix_nat48="fd00:0064:0123::"
  prefix_nat64="fd00:0064:0123:4567::"
  prefix_ula="fdff:cafe:cafe:cafe::"
  # Commissioner Configuration
  nfc_commissioner_enable=true

Note:

For the compatibility with Decawave dw1000 radio edit the channel to 5 and panid to 0xdeca.

2.2.6.7. Connectivity:

NAT64 technology is enabled by default. It is used to enable communication between the IPv6-only Thread network with pure IPv4 LAN network that the Border Router connects to, as is the case in most applications. In that way, Thread nodes are able to connect to IPv4 cloud services. Thread devices will receive a NAT64-prefix and use it to create their own Unique Local Address.

The other connectivity option, which is also enabled by default, is support for the native IPv6 connection. It uses built-in DHCPv6 client on the Border Router which is able to receive prefixes from the Ethernet interface. If the received prefix is shorter than 63 bits, the 64-bit long subnet prefix is created and forwarded to the Thread network. In such situation, devices create one more address based on the forwarded prefix, unlike in a pure NAT64 solution, which allows thread devices to connect from the Internet.

Note:

When dealing with native IPv6 connectivity, make sure to use the DHCPv6 service, and not the popular Stateless Address Autoconfiguration (SLAAC) tool. This autoconfig tool will only provide a 64-bit long prefix that is not sufficient to delegate a new 64-bit long prefix for the Thread network.

Support for Domain Name System (DNS) resolution is also available. This mechanism is used to translate a host name into a corresponding IP address. It can be used either for translating a domain name into its native IPv6 or for obtaining its IPv4 address and returning IPv6 translated with the DNS64 mechanism.

2.2.6.8. Testing:

To test the Border Router, you can use another node with command line interface (CLI) and ping the Google DNS server 8.8.8.8.

- Flash the CLI Binary onto another Nordic kit connected with EVB1000 as per section 3.2.
- Start a terminal emulator like PuTTY/minicom/pyterm/ and connect to the used COM port with the settings described in section 2.2.4
- Run the following commands:
  panid 0xdeca
  channel 5
  ifconfig up
  thread start
  state
  ping fd00:0064:0123:4567::0808:0808

Note:
0808:0808 is in fact the Google DNS server address "8.8.8.8" in hex representation. In that way, you can reach any IPv4 cloud by replacing last 32 bits of an IPv6 address with a correctly encoded IPv4 address.

- After running the command, you should receive the following result:
  16 bytes from fd00:64:123:4567:0:0:808:808: icmp_seq=5 hlim=39 time=111ms

2.2.7. COAP Based Cloud Application:

COAP Based Cloud Application which will send the integer data to the cloud through dw1000 radio and Nordic Host controller with the COAP support in openthread stack.

The setup for this example is same as defined in the thread border router example.

Steps to be followed to test the COAP Based Cloud Application is defined as follows.

1. Register with cloud (things.io)
2. Create a new Project
3. Add the token in the CLI source code and recompile the code.
4. Create a widget on the web interface
5. Testing

Here will explain these steps in detail.

2.2.7.1. Register with cloud

For a detailed description of the sign-up procedure and an introduction to the web interface of the things.io, complete the steps described in the Getting Started web link of www.things.io

2.2.7.2. Create a new Project

- Log in to the main control panel.
- Go to the Things Manager page.
- Create a new product. Select Raspberry Pi as Board
- Choose JSON as the data type.
- Activate the Thing by clicking Activate More Things.
- After the Thing has been activated, copy the related Thing Token.

2.2.7.3. Add the token in the CLI source code and recompile the code.

The Token must be copied into the CLI application source code. i.e., The value can be found in the header of the file coap_api.cpp represented as the CLOUD_URI_PATH define. This file located in openthread-master/src/core/api/ directory. Replace Thing Token ({THING_TOKEN} string) with the appropriate one, obtained in the process of Cloud setup.

```
#define CLOUD_URI_PATH    "v2/things/{THING_TOKEN}"
```

For Example:

```
#define CLOUD_URI_PATH    "v2/things/DFdOKr5AHo_7Aj-L7UNnIO4BSunTvQeaJSgCWILuYA0"
```

- After replacing the token, recompile the code with enabling COAP support as defined below

  "COAP=1 ENABLE_DW1000=1 make -f examples/Makefile-nrf52840-dw1000"

- Flash the CLI on one node and NCP on another node and connect to the Raspberry Pi as mentioned in the "Thread_Border_Router" setup.
- Start the Thread Network as mentioned in the "Thread Border Router Example".
- On CLI node by using "coapsend" command, send the data (first integer) value to the cloud

  "coapsend 45"

2.2.7.4. Create a widget on the web interface

Go to the Dashboard page.

- Remove all temporary created widgets by clicking Edit Dashboard (Optional).
- Click Add Widget (+).
- Fill in the name of the widget (say "Temperature") and then choose Thing Resource as a "Data Source" and temp as a Resource.
- Use Gauge as Widget Type and check the Realtime check box.
- Optionally, you may also fill in the ranges, units (Say "Celsius").

- Now will observe the Gauge widget with Temperature value as 45.

2.2.7.5. Testing:

When we test from the CLI node as defined below continuously using "coapsend" command and then will observe the changes in Temperature Widget on the cloud web interface.

- "coapsend 50"
- "coapsend 30"

3. Openthread Overview

Openthread is an open-source implementation of the Thread networking protocol. Nest has released OpenThread to make the technology used in Nest products more broadly available to developers to accelerate the development of products for the connected home.

Openthread is an OS and platform agnostic, with a narrow platform abstraction layer and a small memory footprint, making it highly portable.

Openthread is Thread Certified Component, implementing all features defined in the Thread 1.1.1 specification. This specification defines an IPv6-based reliable, secure and low-power wireless device-to-device communication protocol for home applications.

3.1. Openthread Features

OpenThread implements all features defined in the Thread 1.1.1 specification, including all Thread networking layers (IPv6, 6LoWPAN, IEEE 802.15.4 with MAC security, Mesh Link Establishment, Mesh Routing) and device roles.

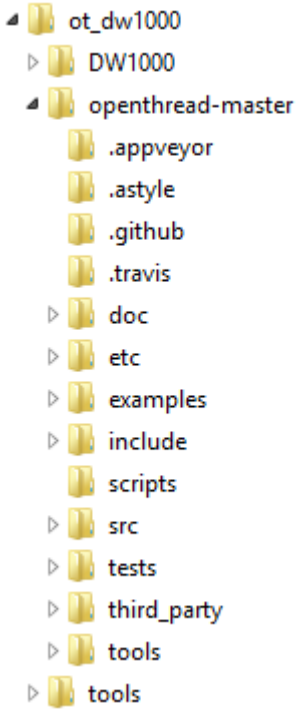
OpenThread supports both system-on-chip (SoC) and network co-processor (NCP) designs. Other features and enhancements include:

- Application support and services
  - IPv6 configuration and raw data interface
  - UDP sockets
  - CoAP client and server
  - DHCPv6 client and server
  - DNSv6 client
  - Command Line Interface (CLI)
- NCP support
  - Spinel - general purpose NCP protocol
  - wpantund - user-space NCP network interface driver/daemon
  - Sniffer support via NCP Spinel nodes

4. OpenThread Code / Directory Organization

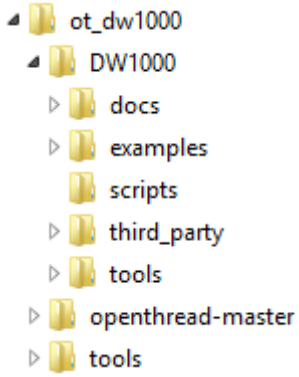
4.1. Top Level

OpenThread source code directory will have following folders. Let's consider 01_00_03 version release as an example



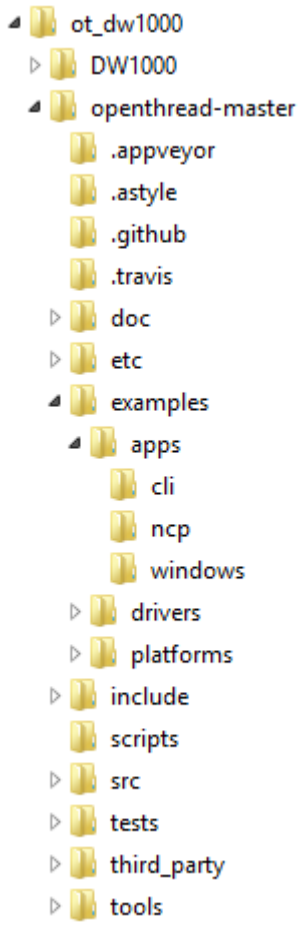
4.2. Docs

The docs folder contains release notes, user guide and test Reports for Open Thread CLI and NCP Application tested for EVB1000 with Nordic NRF52840 platforms.



4.3. Examples

The examples folder is the root folder for Open Thread CLI and NCP Applications


ot_dw1000
DW1000
openthread-master
.appveyor
.astyle
.github
.travis
doc
etc
examples
apps
cli
ncp
windows
drivers
platforms
include
scripts
src
tests
third_party
tools