NXP Semiconductors
User's Guide

Document Number: OpenIL_UG
Rev. 1.3.1, Oct/2018

Open Industrial Linux User Guide

Contents

Chapter 1
Introduction

This document provides a complete description of Open Industrial Linux (OpenIL) features, getting started on OpenIL using NXP OpenIL platforms, and the various software settings involved. It describes in detail the industrial features, which include NETCONF/YANG, TSN, Xenomai, IEEE 1588, OP-TEE, and SELinux. It also includes detailed steps for running the demos such as Selinux demo, 1-board TSN Demo, 3-board TSN demo, 4G-LTE demo, and OTA implementation. It also provides a complete description of the OpenIL compilation steps.

1.1 Acronyms and abbreviations

The following table lists the acronyms used in this document.

Table 1. Acronyms and abbreviations

Term	Description
BC	Boundary clock
BMC	Best master clock
CA	Client application
CAN	Controller Area Network
DEI	Drop eligibility indication
EtherCAT	Ethernet for Control Automation Technology
FMan	Frame manager
ICMP	Internet control message protocol
IETF	Internet engineering task force
IPC	Inter process communication
KM	Key management
LBT	Latency and bandwidth tester
MAC	Medium access control
NMT	Network management
OC	Ordinary clock
OpenIL	Open industry Linux
OP-TEE	Open portable trusted execution environment
OS	Operating system
OTA	Over-the air
OTPMK	One-time programmable master key
PCP	Priority code point

Table continues on the next page...

Table 1. Acronyms and abbreviations (continued)

Term	Description
PDO	Process data object
PHC	PTP hardware clock
PIT	Packet inter-arrival times
PTP	Precision time protocol
QSPI	Queued serial peripheral interface
RCW	Reset configuration word
REE	Rich execution environment
RPC	Remote procedure call
RTT	Round-trip times
SABRE	Smart Application Blueprint for Rapid Engineering
SDO	Service data object
SRK	Single root key
TA	Trusted application
TAS	Time-aware scheduler
TCP	Transmission control protocol
TEE	Trusted execution environment
TFTP	Trivial file transfer protocol
TSN	Time sensitive networking
TZASC	Trust zone address space controller
UDP	User datagram protocol
VLAN	Virtual local area network

1.2 Reference documentation

1. Refer to the following documents for detailed instructions on booting up the NXP hardware boards supported by Open IL:
   - LS1012ARDB Getting Started Guide.
   - LS1021-IoT Getting Started Guide.
   - LS1043ARDB Getting Started Guide.
   - LS1046ARDB Getting Started Guide.
   - i.MX6 SabreSD Board Quick Start Guide
2. For booting up LS1021A-TSN board, refer to the Section Booting up the board on page 16 of this document.
3. For the complete description of the industrial IoT baremetal framework, refer to the latest available version of Industrial IoT Baremetal Framework Developer Guide.

1.3 About OpenIL

The OpenIL project (“Open Industry Linux”) is designed for embedded industrial usage. It is an integrated Linux distribution for industry.

OpenIL is built on buildroot project and provides packages for the industrial market.

- **Focus on industry**: OpenIL provides key components for industry usage, for example, Time sensitive network (TSN), Netconf, IEEE 1588, and Xenomai.
- **Ease of use**: OpenIL is a tool that simplifies and automates the process of building a complete Linux system for an embedded system, using cross-compilation. It follows the buildroot project rules. For more buildroot information, refer to the page: https://buildroot.org/
- **Extensibility**: OpenIL provides capabilities of industry usage and standardized Linux system packages. And user can also easily replicate the same setup on customized packages and devices.
- **Lightweight**: OpenIL only includes necessary Linux packages and industry packages in order to make the system more lightweight to adapt to industry usage. Users can customize the package via a configuration file.
- **Open Source**: OpenIL is an open project. Anyone can participate in the OpenIL development through the Open Source community.

1.3.1 OpenIL Organization

OpenIL follows the Buildroot directory structure depicted in the following figure. The second and third levels of the directory are generated during compilation.

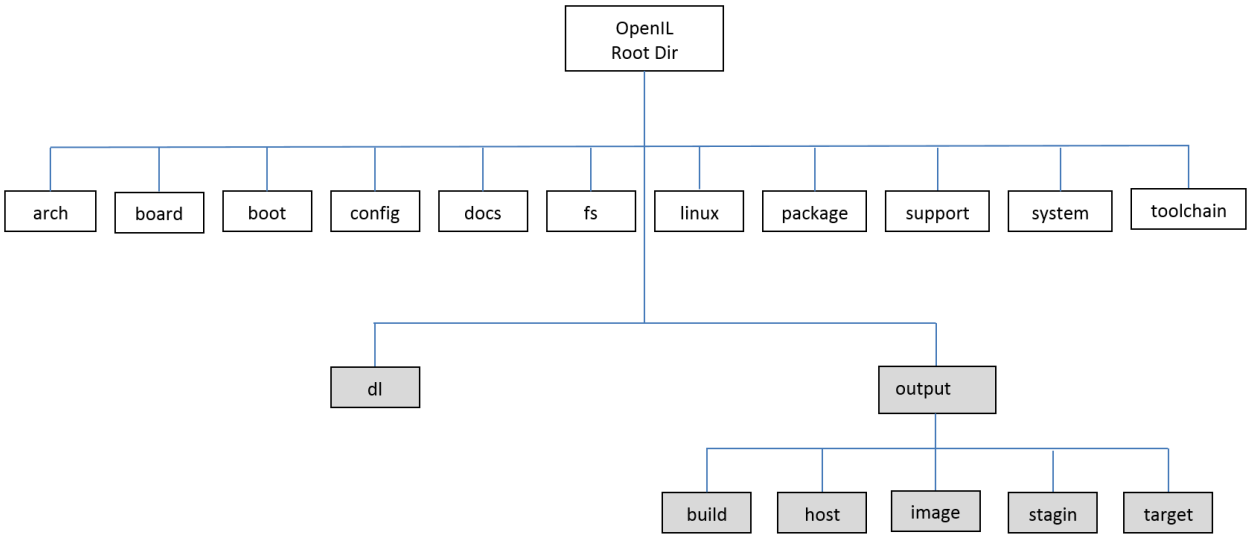


Figure 1. OpenIL structure

Table 2. Source directories

Directory name	Description
arch	Files defining the architecture variants (processor type, ABI, floating point, etc.)
toolchain	Packages for generating or using tool-chains
system	Contains the rootfs skeleton and options for system-wide features

Table continues on the next page...

Table 2. Source directories (continued)

Directory name	Description
linux	The linux kernel package.
package	All the user space packages (1800+)
fs	Logic to generate file system images in various formats
boot	Boot-loader packages
configs	Default configuration files for various platforms
board	Board-specific files (kernel configurations, patches, image flashing scripts, etc.)
support	Miscellaneous utilities (kconfig code, libtool patches, download helpers, and more)
docs	Documentation

Table 3. Build directories

Directory name	Description
dl	Path where all the source tarballs are downloaded
output	Global output directory
output/build	Path where all source tarballs are extracted and the build of each package takes place.
output/host	Contains both the tools built for the host and the sysroot of the toolchain
output/staging	A symbolic link to the sysroot, that is, to host/<tuple>/sysroot/ for convenience
output/target	The target Linux root filesystem, used to generate the final root filesystem images
output/images	Contains all the final images: kernel, bootloader, root file system, and so on

1.3.2 Host system requirements

OpenIL is designed to build in Linux systems. The following host environments have been verified to build the OpenIL.

- CentOS Linux 7 (Core)
- CentOS release 6.7 (Final)
- Ubuntu 16.10
- Ubuntu 16.04
- Ubuntu 14.04

While OpenIL itself builds most host packages it needs for the compilation, certain standard Linux utilities are expected to be already installed on the host system. The following tables provide an overview of the mandatory and optional packages.

NOTE

Package names listed in the following tables might vary between distributions.

Table 4. Host system mandatory packages

Mandatory packages	Note
which	
sed	
make	Version 3.81 or later
binutils	
build-essential	Only for Debian based systems
gcc	Version 2.95 or later
g++	Version 2.95 or later
bash	
patch	
gzip	
bzip2	
perl	Version 5.8.7 or later
tar	
cpio	
python	Version 2.6 or later
unzip	
rsync	
file	Must be in /usr/bin/file
bc	
wget	
autoconf, dh-autoreconf	
openssl, libssl-dev	
libmagickwand-dev (Debian, Ubuntu) imageMagick-devel (CentOS)	

Table 5. Host system optional packages

Optional packages	Note
ncurses5	To use the menuconfig interface
qt4	To use the xconfig interface
glib2, gtk2 and glade2	To use the gconfig interface

Table continues on the next page...

Table 5. Host system optional packages (continued)

Optional packages	Note
bazaar	Source fetching tools. If you enable packages using any of these methods, you need to install the corresponding tool on the host system
cvs	
git	
mercurial	
scp	
javac compiler	Java-related packages, if the Java Classpath needs to be built for the target system
jar tool	
asciidoc	Documentation generation tools
w3m	
python with the argparse module	
dblatex	
graphviz	To use graph-depends and <pkg>-graph-depends
python-matplotlib	To use graph-build

1.4 Feature set summary

This section provides a summary of OpenIL's compilation and industrial features.

1.4.1 Compilation features

The following are the compilation features:

- **Specify partition size** of the storage for the filesystem by using the `make menuconfig` command.

```
System configuration --->
(300M) Partition size of the storage for the rootfs
```

  This configuration specifies the size of the storage device partition for the building rootfs and currently used by NXP platforms and SD card device. To set the size of the partition with `300M`, `2G` or other values, the target system can get the specific size of partition space for the using filesystem.

- **Support custom filesystem** (that is, Ubuntu)

  Users can download OpenIL and build the target system with an Ubuntu filesystem. The specific filesystem can be set conveniently by using the `make menuconfig` command.

```
System configuration --->
   Root FS skeleton (custom target skeleton) --->
        Custom skeleton via network --->
```

  Currently, there are four NXP platforms that can support Ubuntu filesystem:

  - `configs/nxp_ls1043ardb-64b_ubuntu_defconfig`

- `configs/nxp_ls1046ardb-64b_ubuntu_defconfig`
- `configs/nxp_ls1021aiot_ubuntu_defconfig`
- `configs/imx6q-sabresd_ubuntu_defconfig`

1.4.2 Supported industrial features

The following are the industrial features supported by OpenIL:

- Netconf/Yang
- Netopeer
- TSN
- IEEE 1588
- IEEE 1588 2-step E2E transparent clock support
- Xenomai Cobalt mode
- SELinux (Ubuntu)
- OP-TEE
- DM-Crypt
- Baremetal

These are explained in detail in Industrial features on page 27.

NOTE

For the complete description of the Industrial IoT baremetal framework, refer to the document,*Industrial_IoT_Baremetal_Framework_Developer_Guide*.

1.5 Supported NXP platforms

The following table lists the NXP platforms supported by OpenIL.

Table 6. Supported NXP platforms

Platform	Architecture	Configuration file in OpenIL	Boot
ls1021atsn	ARM v7	configs/nxp_ls1021atsn_defconfig	SD
ls1021atsn (OP-TEE)	ARM v7	configs/nxp_ls1021atsn_optee-sb_defconfig	SD
ls1021aiot	ARM v7	configs/nxp_ls1021aiot_defconfig	SD
ls1021aiot (OP-TEE)	ARM v7	configs/nxp_ls1021aiot_optee_defconfig	SD
ls1043ardb (32-bit)	ARM v8	configs/nxp_ls1043ardb-32b_defconfig	SD
ls1043ardb (64-bit)	ARM v8	configs/nxp_ls1043ardb-64b_defconfig	SD
ls1043ardb (Ubuntu)	ARM v8	configs/nxp_ls1043ardb-64b_ubuntu_defconfig	SD
ls1046ardb (32-bit)	ARM v8	configs/nxp_ls1046ardb-32b_defconfig	SD
ls1046ardb (64-bit)	ARM v8	configs/nxp_ls1046ardb-64b_defconfig	SD

Table continues on the next page...

Table 6. Supported NXP platforms (continued)

Platform	Architecture	Configuration file in OpenIL	Boot
ls1046ardb (Ubuntu)	ARM v8	configs/nxp_ls1046ardb-64b_ubuntu_defconfig	SD
ls1012ardb (32-bit)	ARM v8	configs/nxp_ls1012ardb-32b_defconfig	QSPI
ls1012ardb (64-bit)	ARM v8	configs/nxp_ls1012ardb-64b_defconfig	QSPI
i.MX6Q SabreSD	ARM v7	configs/imx6q-sabresd_defconfig	SD
i.MX6Q SabreSD	ARM v7	configs/imx6q-sabresd_ubuntu_defconfig	SD

1.5.1 Default compilation settings for NXP platforms

The following table provides the default compilation settings for each OpenIL NXP platform.

Table 7. Default compilation settings

Platform	Toolchain	libc	Init system	Filesystem
ls1021atsn	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1021atsn (OP-TEE)	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1021aiot	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1021aiot (OP-TEE)	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1043ardb (32-bit)	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1043ardb (64-bit)	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1043ardb (Ubuntu)	gcc 5.x	glibc 2.23	Systemd	ubuntu-base-16.04.3-arm64
ls1046ardb (32-bit)	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1046ardb (64-bit)	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1046ardb (Ubuntu)	gcc 5.x	glibc 2.23	Systemd	ubuntu-base-16.04.3-arm64
ls1012ardb (32-bit)	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1012ardb (64-bit)	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
i.MX6Q SabreSD	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
i.MX6Q SabreSD	gcc 5.x	glibc 2.23	Systemd	ubuntu-base-16.04.5-arm

Chapter 2
Getting started

After reading this section, you should be able to get the OpenIL source code, build and program the NXP platform images, and run the OpenIL system on the supported NXP platforms.

2.1 Getting OpenIL

OpenIL releases are available every few months. The Release Number follows the format 'YYYYMM', for example, 201708. Release tarballs are available at: https://github.com/openil/openil.

To follow development, make a clone of the Git repository. Use the below command:

```
$ git clone https://github.com/openil/openil.git
$ cd openil
# checkout to the 201808 v1.3.1 release
$ git checkout OpenIL-201810 -b OpenIL-201810
```

2.2 OpenIL quick start

The steps below help the user to build the NXP platform images with OpenIL quickly. Ensure to follow the important notes provided in the following section.

2.2.1 Important notes

- Build everything as a normal user. There is no need to be a root user to configure and use OpenIL. By running all commands as a regular user, you protect your system against packages behaving badly during compilation and installation.
- Do not use `make -jN` command to build OpenIL as the top-level parallel `make` is currently not supported.
- The `PERL_MM_OPT` issue: You might encounter an error message for the `PERL_MM_OPT` parameter when using the `make` command in some host Linux environment as shown below:

  ```
  You have PERL_MM_OPT defined because Perl local::lib is installed on your system.
  Please unset this variable before starting Buildroot, otherwise the compilation of Perl
  related packages will fail.
      make[1]: *** [core-dependencies] Error 1
      make: *** [_all] Error 2
  ```

  To resolve this issue, just unset the `PERL_MM_OPT` parameter.

  ```
  $ unset PERL_MM_OPT
  ```

2.2.2 Building the final images

For the NXP platforms supported by OpenIL, the default configuration files can be found in the `configs` directory. The following table describes the default configuation files for the NXP-supported OpenIL platforms.

Table 8. Default configuration

Platform	Configuration file in OpenIL
ls1021atsn	configs/nxp_ls1021atsn_defconfig
ls1021atsn (OP-TEE)	configs/nxp_ls1021atsn_optee-sb_defconfig
ls1021aiot	configs/nxp_ls1021aiot_defconfig
ls1021aiot (OP-TEE)	configs/nxp_ls1021aiot_optee_defconfig
ls1043ardb (32-bit)	configs/nxp_ls1043ardb-32b_defconfig
ls1043ardb (64-bit)	configs/nxp_ls1043ardb-64b_defconfig
ls1043ardb (Ubuntu)	configs/nxp_ls1043ardb-64b_ubuntu_defconfig
ls1046ardb (32-bit)	configs/nxp_ls1046ardb-32b_defconfig
ls1046ardb (64-bit)	configs/nxp_ls1046ardb-64b_defconfig
ls1046ardb (Ubuntu)	configs/nxp_ls1046ardb-64b_ubuntu_defconfig
ls1012ardb (32-bit)	configs/nxp_ls1012ardb-32b_defconfig
ls1012ardb (64-bit)	configs/nxp_ls1012ardb-64b_defconfig
i.MX6Q SabreSD	configs/imx6q-sabresd_defconfig
i.MX6Q SabreSD (Ubuntu)	configs/imx6q-sabresd_ubuntu_defconfig

The "`configs/nxp_xxxx_defconfig`" files listed in the preceding table include all the necessary U-Boot, Kernel configurations, and application packages for the filesystem. Based on the files without any changes, you can build a complete Linux environment for the target platforms.

To build the final images for an NXP platform (for example, LS1021ATSN), run the following commands:

```
$ cd openil
$ make nxp_ls1021atsn_defconfig
$ make
# or make with a log
$ make 2>&1 | tee build.log
```

NOTE

The `make clean` command should be implemented before any other new compilation.

The `make` command generally performs the following steps:

- Downloads source files (as required and at the first instance);
- Configures, builds, and installs the cross-compilation toolchain;
- Configures, builds, and installs selected target packages;
- Builds a kernel image, if selected;
- Builds a bootloader image, if selected;
- Creates a root filesystem in selected formats.

After the correct compilation, you can find all the images for the platform at `output/images`.

```
images/
├── boot.vfat
```

```
├── ls1021a-tsn.dtb              --- dtb file for ls1021atsn
├── rootfs.ext2
├── rootfs.ext2.gz
├── rootfs.ext2.gz.uboot          --- ramdisk can be used for debug
├── rootfs.ext4.gz -> rootfs.ext2.gz
├── rootfs.tar
├── sdcard.img                    --- entire image can be programmed into the SD
├── uboot-env.bin
├── u-boot-with-spl-pbl.bin      --- uboot image for ls1021atsn
└── uImage                        --- kernel image for ls1021atsn
```

2.3 Booting up the board

Before proceeding further with the instructions in this section, refer to the *Getting Started Guide* of the respective board for detailed instructions regarding board boot-up. See Reference documentation on page 7.

NOTE

- Before booting up the board, you need to install mbed Windows serial port driver in order to obtain the board console. This is a one time activity. Please ignore this step if you have already installed the mbed driver on your system (PC or laptop). You can download the mbed Windows serial port driver from the link below: https://developer.mbed.org/handbook/Windows-serial-configuration.
- Download and install Tera Term on the host computer from the Internet. After installation, a shortcut to the tool is created on the desktop of the host computer.
- If you are using a Windows 10 machine as a host computer and encountering a serial port unstable issue, then, disable the *Volume Storage* service of the Windows machine.

All the NXP platforms can be booted up from the SD card or QSPI flash. After the compilation for one platform, the image files (sdcard.img or qspi.img) are generated in the folder `output/images`. The following table describes the software settings to be used while booting up the NXP platforms with the images built from OpenIL.

Table 9. Switch settings for the board

Platform	Boot	Final image	Board software setting (ON = 1)
LS1021A-TSN	SD card	sdcard.img	SW2 = 0b'111111
LS1021A-IOT	SD card	sdcard.img	SW2[1] = 0b'0
LS1043ARDB	SD card	sdcard.img	SW4[1-8] +SW5[1] = 0b'00100000_0
LS1046ARDB	SD card	sdcard.img	SW5[1-8] +SW4[1] = 0b'00100000_0
LS1012ARDB	QSPI	qspi.img	SW1 = 0b'10100110 SW2 = 0b'00000000
i.MX6Q SabreSD	SD card	sdcard.img	SW6 = 0b'01000010

The flash image (sdcard.img/qspi.img) includes all the RCW, DTB, U-Boot, kernel, Rootfs, and necessary applications.

NOTE

Make sure the board is set to boot up from SD card or QSPI using software configuration. Refer to the preceding table for the switch settings for the respective platform.

.

2.3.1 SD card bootup

For platforms that can be booted up from an SD card, following are the steps to program the sdcard.img.into an SD card:

1. Insert one SD card (at least 2G size) into any Linux host machine.
2. Run the below commands:

```
$ sudo dd if=./sdcard.img of=/dev/sdx
# or in some other host machine:
$ sudo dd if=./sdcard.img of=/dev/mmcblkx

# find the right SD Card device name in your host machine and replace the "sdx" or "mmcblkx".
```

3. Now, insert the SD card into the target board (switch the board boot from SD card first) and power on.

2.3.2 QSPI bootup

For platforms that can be booted up from QSPI (for example, LS1012ARDB), following are the steps to program the qspi.img into QSPI flash.

Set the board boot from QSPI, then power on, and enter the U-Boot command environment.

```
=>i2c mw 0x24 0x7 0xfc; i2c mw 0x24 0x3 0xf5
=>tftp 0x80000000 qspi.img
=>sf probe 0:0
=>sf erase 0x0 +$filesize
=>sf write 0x80000000 0x0 $filesize
=>reset
```

2.3.3 Starting up the board

After the sdcard.img/qspi.img programming, startup the board. You should see the following information.

```
Starting logging: OK
Initializing random number generator... [    6.120727] random: dd urandom read with 13 bits of entropy available
done.
Mounting cgroupfs hierarchy: OK
Starting system message bus: done
Starting network: OK
ssh-keygen: generating new host keys: RSA DSA ECDSA ED25519
Starting sshd: OK
resize2fs 1.43.3 (04-Sep-2016)
Filesystem at /d[    7.560413] EXT4-fs (mmcblk0p2): resizing filesystem from 212800 to 2097152 blocks
ev/mmcblk0p2 is [    7.568527] EXT4-fs (mmcblk0p2): Converting file system to meta_bg
mounted on /; on-line resizing required
old_desc_blocks = 1, new_desc_blocks = 8
[    7.611678] EXT4-fs (mmcblk0p2): resized filesystem to 2097152
[    7.825614] random: nonblocking pool is initialized
The filesystem on /dev/mmcblk0p2 is now 2097152 (1k) blocks long.


       _______
      |       |  Open Industrial Linux          |  |
      |       |   _______   _______   _____     |  |
      |       |  |       | |       | |     |    |  |
      |       |  |       | |_______| |     |    |  |
      |       |  |       |  _______  |     |    |  |
      |_______|  |_______| |_______  |     |    |  |_____
                 |                                openil.org


[root@OpenIL:~]# _
```

Figure 2. OpenIL system startup

The system will be logged in automatically.

2.4 Basic OpenIL operations

This section describes the commands that can be used for performing basic OpenIL operations.

Sample usages of the 'make' command:

- Displays all commands executed by using the make command:

```
$ make V=1 <target>
```

- Displays the list of boards with a defconfig:

```
$ make list-defconfigs
```

- Displays all available targets:

```
$ make help
```

- Sets Linux configurations:

```
$ make linux-menuconfig
```

- Deletes all build products (including build directories, host, staging and target trees, images, and the toolchain):

```
$ make clean
```

- Resets OpenIL for a new target.

- Deletes all build products as well as the configuration (including `dl` directory):

```
$ make distclean
```

NOTE

Explicit cleaning is required when any of the architecture or toolchain configuration options are changed.

- **Downloading, building, modifying, and rebuilding a package**

  Run the below command to build and install a particular package and its dependencies:

```
$ make <pkg>
```

  For packages relying on the OpenIL infrastructure, there are numerous special `make` targets that can be called independently such as the below command:

```
$ make <pkg>-<target>
```

The package build targets are listed in the following table.

Table 10. Package build targets

Package Target	Description
<pkg>	Builds and installs a package and all its dependencies

Table continues on the next page...

Table 10. Package build targets (continued)

Package Target	Description
<pkg>-source	Downloads only the source files for the package
<pkg>-extract	Extracts package sources
<pkg>-patch	Applies patches to the package
<pkg>-depends	Builds package dependencies
<pkg>-configure	Builds a package up to the configure step
<pkg>-build	Builds a package up to the build step
<pkg>-show-depends	Lists packages on which the package depends
<pkg>-show-rdepends	Lists packages which have the package as a dependency
<pkg>-graph-depends	Generates a graph of the package dependencies
<pkg>-graph-rdepends	Generates a graph of the package's reverse dependencies
<pkg>-dirclean	Removes the package's build directory
<pkg>-reconfigure	Restarts the build from the configure step
<pkg>-rebuild	Restarts the build from the build step

Thus, a package can be downloaded in the directory `dl/`, extracted to the directory `output/build/<pkg>`, and then built in the directory `output/build/<pkg>`. You need to modify the code in the `output/build/<pkg>`, and then run the command, `$make <pkg>-rebuild` to rebuild the package.

For more details about OpenIL operations, refer to the Buildroot document available at the URL: https://buildroot.org/downloads/manual/manual.html#getting-buildroot.

Chapter 3
NXP OpenIL platforms

OpenIL supports the following NXP Layerscape ARM® platforms: LS1012ARDB, LS1021A-TSN, LS1021-IoT, LS1043ARDB, LS1046ARDB, and i.MX6QSabreSD. For more information about those platforms, refer to the following URLs:

- http://www.nxp.com/products/microcontrollers-and-processors/arm-processors/qoriq-layerscape-arm-processors:QORIQ-ARM.
- https://www.nxp.com/products/processors-and-microcontrollers/arm-based-processors-and-mcus/i.mx-applications-processors:IMX_HOME

3.1 Introduction

This chapter provides instructions on booting up the boards with a complete SD card or QSPI image. It also describes the process for deploying the U-Boot, Linux kernel, and root file system on the board. The instructions start with generic host and target board pre-requisites. These are followed by the board-specifc configurations listed below:

- Switch settings
- U-Boot environment variables
- Device microcodes
- Reset configuration word (RCW)
- Flash bank usage

NOTE

This chapter is meant for those who want to perform more sub-system debugs, such as U-Boot, kernel, and so on. At the beginning, the board should be booted up and run in U-Boot command environment.

3.2 LS1021A-TSN

The LS1021A Time-Sensitive Networking (TSN) reference design is a platform that allows developers to design solutions with the new IEEE Time-Sensitive Networking (TSN) standard. The board includes the QorIQ Layerscape LS1021A industrial applications processor and the SJA1105T TSN switch. The LS1021A-TSN is supported by an industrial Linux SDK with Xenomai real time Linux, which also provides utilities for configuring TSN on the SJA1105T switch.

With virtualization support, trust architecture, secure platform, Gigabit Ethernet, SATA interface, and an Arduino Shield connector for multiple wireless modules, the LS1021A-TSN platform readily supports industrial IoT requirements.

3.2.1 Switch settings

The following table lists and describes the switch configuration for LS1021ATSN board.

NOTE

OpenIL supports only the SD card boot for LS1021ATSN platform.

Table 11. LS1021ATSN SD boot software setting

Platform	Boot source	Software setting
LS1021ATSN	SD card	SW2 = 0b'111111

3.2.2 Updating target images

Use the following commands to build the images for LS1021A-TSN platform:

- **Building images**

```
$ cd openil
$ make nxp_ls1021atsn_defconfig
$ make
# or make with a log
$ make 2>&1 | tee build.log
```

- **Programming U-Boot with RCW in SD card**

  Power on the LS1021A-TSN board to the U-Boot command environment, then use the following commands:

```
=>tftp 81000000 u-boot-with-spl-pbl.bin
=>mmc erase 8 0x500
=>mmc write 0x81000000 8 0x500
#then reset the board
```

- **Deploying kernel and Ramdisk from TFTP**

  1. Set the U-Boot environment.

```
=>setenv bootargs 'root=/dev/ram0 rw ramdisk_size=50000000 console=ttyS0,115200'
=>saveenv
```

  2. Boot up the system.

```
=>tftp 83000000 uImage
=>tftp 88000000 rootfs.ext2.gz.uboot
=>tftp 8f000000 ls1021a-tsn.dtb
=>bootm 83000000 88000000 8f000000
```

3.3 LS1021A-IoT

The LS1021A-IoT gateway reference design is a purpose-built, small footprint hardware platform equipped with a wide array of both high-speed connectivity and low speed serial interfaces. It is engineered to support the secure delivery of IoT services to end-users at their home, business, or other commercial locations. The LS1021A-IoT gateway reference design offers an affordable, ready-made platform for rapidly deploying a secure, standardized, and open infrastructure gateway platform for deployment of IoT services.

3.3.1 Switch settings

The following table lists and describes the switch configuration for LS1021A-IoT board.

NOTE

OpenIL supports only the SD card boot for the LS1021A-IoT platform.

Table 12. LS1021A-IoT SD boot software setting

Platform	Boot source	software setting
LS1021A-IoT	SD card	SW2[1] = 0b'0

3.3.2 Updating target images

Use the following commands to build the images for LS1021A-IoT platform:

- **Building images**

```
$ cd openil
$ make nxp_ls1021aiot_defconfig
$ make
# or make with a log
$ make 2>&1 | tee build.log
```

- **Programming U-Boot with RCW on the SD card**

  Power on the LS1021A-IoT board to U-Boot command environment. Then, use the commands below:

```
=>tftp 81000000 u-boot-with-spl-pbl.bin
=>mmc erase 8 0x500
=>mmc write 0x81000000 8 0x500
#then reset the board
```

- **Deploying kernel and Ramdisk from TFTP**

  1. Set the U-Boot environment.

```
=>setenv bootargs 'root=/dev/ram0 rw ramdisk_size=50000000 console=ttyS0,115200'
=>saveenv
```

  2. Boot up the system.

```
=>tftp 83000000 uImage
=>tftp 88000000 rootfs.ext2.gz.uboot
=>tftp 8f000000 ls1021a-iot.dtb
=>bootm 83000000 88000000 8f000000
```

3.4 LS1043ARDB and LS1046ARDB

The QorIQ LS1043A and LS1046A reference design boards are designed to exercise most capabilities of the LS1043A and LS1046A devices. These are NXP's first quad-core, 64-bit ARM®-based processors for embedded networking and industrial infrastructure.

3.4.1 Switch settings

OpenIL supports only the SD card boot mode for LS1043ARDB and the LS1046ARDB platforms.

Table 13. LS1043ARDB/LS1046ARDB SD boot software settings

Platform	Boot source	Software setting
LS1043ARDB	SD card	SW4[1-8] +SW5[1] = 0b'00100000_0
LS1046ARDB	SD card	SW5[1-8] +SW4[1] = 0b'00100000_0

NOTE

In order to identify the LS1043A silicon correctly, users should ensure that the SW5[7-8] is = 0b’11.

3.4.2 Updating target images

For LS1043ARDB and LS1046ARDB platforms, the OpenIL can support 32-bit and 64-bit systems. Use the following commands to build the images for the LS1043ARDB or LS1046ARDB platforms:

- **Building images**

```
$ cd openil
$ make nxp_ls1043ardb-32b_defconfig
# or
$ make nxp_ls1046ardb-32b_defconfig
$ make
# or make with a log
$ make 2>&1 | tee build.log
```

- **Programming U-Boot with RCW, FMan ucode, and PPA in SD card**

  Power on the LS1043ARDB / LS1046ARDB board to U-Boot command environment, then use the following commands:

```
# programming U-Boot with RCW
=> tftpboot 82000000 u-boot-with-spl-pbl.bin
=> mmc write 82000000 8 800
# programming the FMan ucode
=> tftpboot 82000000 fsl_fman_ucode_ls1043_r1.0_108_4_5.bin
=> mmc write 82000000 820 50
# programming the PPA firmware
=> tftpboot 82000000 ppa.itb
=> mmc write 82000000 2800 36

#then reset the board
```

- **Deploying kernel and Ramdisk from TFTP**

  1. Set the U-Boot environment.

```
=>setenv bootargs "root=/dev/ram0 earlycon=uart8250,mmio,0x21c0500 console=ttyS0,115200"
=>saveenv
```

  2. Boot up the system.

```
# for ls1043ardb
=>tftp a0000000 kernel-ls1043a-rdb-aarch32.itb
# for ls1046ardb
=>tftp a0000000 kernel-ls1046a-rdb-aarch32.itb

=>bootm a0000000
```

3.5 LS1012ARDB

The QorIQ LS1012A processor delivers enterprise-class performance and security capabilities to consumer and networking applications in a package size normally associated with microcontrollers. Combining a 64-bit ARM®v8-based processor with network packet acceleration and QorIQ trust architecture security capabilities, the LS1012A features line-rate networking performance at 1 W typical power in a 9.6 mm x 9.6 mm package.

The QorIQ LS1012A reference design board (LS1012A-RDB) is a compact form-factor tool for evaluating LS1012A application solutions. The LS1012A-RDB provides an Arduino shield expansion connector for easy prototyping of additional components such as an NXP NFC Reader module.

3.5.1 Switch settings

The LS1012ARDB platform can be booted up only using the QSPI Flash.

The table below lists the default switch settings and the description of these settings.

Table 14. LS1012ARDB QSPI boot software settings

Platform	Boot source	SW setting
LS1012ARDB	QSPI Flash 1	SW1 = 0b'10100110 SW2 = 0b'00000000
	QSPI Flash 2	SW1 = 0b'10100110 SW2 = 0b'00000010

3.5.2 Updating target images

For LS1012ARDB platform, the OpenIL supports 32-bit and 64-bit systems. Use the following commands to build the images for the LS1012ARDB platform:

- **Building images**

```
$ cd openil
$ make nxp_ls1012ardb-32b_defconfig
$ make
# or make with a log
$ make 2>&1 | tee build.log
```

- **Programming U-Boot, RCW, and PPA in QSPI**

  Power on the LS1012ARDB board to U-Boot command environment. Then, use the commands below:

```
 # programming U-Boot
=>i2c mw 0x24 0x7 0xfc; i2c mw 0x24 0x3 0xf5
=>tftp 0x80000000 <u-boot_file_name>.bin
=>sf probe 0:0
=>sf erase 0x100000 +$filesize
=>sf write 0x80000000 0x100000 $filesize
 # programming RCW
=>i2c mw 0x24 0x7 0xfc; i2c mw 0x24 0x3 0xf5
=>tftp 0x80000000 <rcw_file_name>.bin
=>sf probe 0:0
=>sf erase 0x0 +$filesize
```

```
=>sf write 0x80000000 0x0 $filesize
 # programming PPA
=> tftp 0x80000000 ppa.itb
=> sf probe 0:0
=> sf erase 0x500000 +$filesize
=> sf write 0x80000000 0x500000 $filesize
 # then reset the board
```

- **Deploying kernel and RAMdisk from TFTP**

  1. Set the U-Boot environment.

     ```
     =>setenv bootargs 'ttyS0,115200 root=/dev/ram0 earlycon=uart8250,mmio,0x21c0500'
     =>saveenv
     ```

  2. Boot up the system.

     ```
     =>tftp a0000000 kernel-ls1012a-rdb-aarch32.itb
     =>bootm a0000000
     ```

3.6 i.MX6QSabreSD

The i.MX 6Dual/6Quad processors feature NXP's advanced implementation of the quad ARM® Cortex®-A9 core, which operates at speeds up to 1 GHz. These processors include 2D and 3D graphics processors, 3D 1080p video processing, and integrated power management. Each processor provides a 64-bit DDR3/LVDDR3/LPDDR2-1066 memory interface and a number of other interfaces for connecting peripherals, such as WLAN, Bluetooth®, GPS, hard drive, displays, and camera sensors.

The Smart Application Blueprint for Rapid Engineering (SABRE) board for smart devices introduces developers to the i.MX 6 series of applications processors. Designed for ultimate scalability, this entry level development system ships with the i.MX 6Quad applications processor but is schematically compatible with i.MX6 Dual, i.MX6 DualLite, and i.MX6 Solo application processors. This helps to reduce time to market by providing a foundational product design and serves as a launching point for more complex designs.

3.6.1 Switch settings for the i.MX6Q SabreSD

The following table lists and describes the switch configuration for i.MX6Q SabreSD board:

NOTE

OpenIL supports only the SD card boot for the i.MX6Q SabreSD platform.

Table 15. Switch configuration for the i.MX6Q SabreSD board

Platform	Boot source	Software setting
i.MX6Q SabreSD	SD card on slot 3	SW2[1] = 0b’01000010

3.6.2 Updating target images

Use the following commands to build the images for i.MX6Q SabreSD platform:

Building images

```
$ cd openil
$ make imx6q-sabresd_defconfig
$ make
```

```
# or make with a log
$ make 2>&1 | tee build.log

# See built images as follows:
$ ls output/images/
boot.vfat  imx6q-sabresd.dtb  rootfs.ext2  rootfs.ext2.gz  rootfs.ext4.gz  rootfs.tar  sdcard.img
SPL  u-boot.bin  u-boot.img  zImage
```

Programming U-Boot on the SD card

Power on the board to U-Boot command environment. Then, use the commands below:

```
$ dd if=SPL of=/dev/sdX bs=1K seek=1
$ dd if=u-boot.imx of=/dev/sdX bs=1K seek=69; sync
```

NOTE

Replace `sdX` with your own SD card 'node name' detected by the system.

Deploying kernel and device tree image

Kernel and device tree image are stored in the first partition (vfat) of SD card.

```
$ cp -avf imx6q-sabresd.dtb /mnt
$ cp -avf zImage /mnt
$ umount /mnt
```

NOTE

`/mnt` is the mount point of the vfat partition.

Chapter 4
Industrial features

This section provides a description of the following industrial features: NETCONF/YANG, TSN, Xenomai, IEEE 1588, OP-TEE, and SELinux.

NOTE

The Industrial IoT baremetal framework is described in the document, *Industrial_IoT_Baremetal_Framework_Developer_Guide*.

4.1 NETCONF/YANG

- NETCONF v1.0 and v1.1 compliant (RFC 6241)
- NETCONF over SSH (RFC 6242) including Chunked Framing Mechanism
- DNSSEC SSH Key Fingerprints (RFC 4255)
- NETCONF over TLS (RFC 5539bis)
- NETCONF Writable-running capability (RFC 6241)
- NETCONF Candidate configuration capability (RFC 6241)
- NETCONF Validate capability (RFC 6241)
- NETCONF Distinct startup capability (RFC 6241)
- NETCONF URL capability (RFC 6241)
- NETCONF Event Notifications (RFC 5277 and RFC 6470)
- NETCONF With-defaults capability (RFC 6243)
- NETCONF Access Control (RFC 6536)
- NETCONF Call Home (Reverse SSH draft, RFC 5539bis)
- NETCONF Server Configuration (IETF Draft)

4.2 TSN

On the LS1021A-TSN platform, TSN features are implemented as part of the **SJA1105TEL** Automotive Ethernet L2 switch. These are:

- MII, RMII, RGMII, 10/100/1000 Mbps
- IEEE 802.1Q: VLAN frames and L2 QoS
- IEEE 1588v2: Hardware forwarding for one-step sync messages
- IEEE 802.1Qci: Ingress rate limiting (per-stream policing)
- IEEE 802.1Qbv: Time-aware traffic shaping
- Statistics for transmitted, received, dropped frames, buffer load
- TTEthernet (SAE AS6802)

4.3 Xenomai

Xenomai is a free software framework adding real-time capabilities to the mainline Linux kernel. Xenomai also provides emulators of traditional RTOS APIs, such as VxWorks® and pSOS®. Xenomai has a strong focus on embedded systems, although it runs over mainline desktop and server architectures as well.

Xenomai 3 is the new architecture of the Xenomai real-time framework, which can run seamlessly side-by-side Linux as a co-kernel system, or natively over mainline Linux kernels. In the latter case, the mainline kernel can be supplemented by the PREEMPT-RT patch to meet stricter response time requirements than standard kernel preemption would bring.

One of the two available real-time cores is selected at build time.

Xenomai can help you in:

- Designing, developing, and running a real-time application on Linux.
- Migrating an application from a proprietary RTOS to Linux.
- Optimally running real-time applications alongside regular Linux applications.

Xenomai features are supported for LS1021A-TSN, LS1043ARDB, LS1046ARDB, and i.MX6Q SabreSD. More information can be found at the Xenomai official website: http://xenomai.org/.

4.3.1 Xenomai running mode

The dual kernel core is codenamed Cobalt, whereas the native Linux implementation is called Mercury. Both Mercury and Cobalt are supported.

4.3.1.1 Running Xenomai Mercury

Xenomai Mercury provides the following API references:

1. Test programs:

- latency: The user manual for Xenomai timer latency benchmark can be found at:

  http://www.xenomai.org/documentation/xenomai-3/html/man1/latency/index.html.

- cyclictest: The user manual for Xenomai high resolution timer test can be found at:

  http://www.xenomai.org/documentation/xenomai-2.6/html/cyclictest/index.html.

2. Utilities:

- xeno: The user manual for Wrapper for Xenomai executables can be found at:

  http://www.xenomai.org/documentation/xenomai-2.6/html/xeno/index.html.

- xeno-config: The user manual for displaying Xenomai libraries configuration can be found at:

  http://www.xenomai.org/documentation/xenomai-2.6/html/xeno-config/index.html.

4.3.1.2 Running Cobalt mode

Xenomai Cobalt provides many APIs to perform testing.

1. Clocktest : The test program `clocktest` provided by Xenomai can be used to test timer APIs. There are three kinds of timer sources: CLOCK_REALTIME, CLOCK_MONOTONIC, and CLOCK_HOST_REALTIME.

- Use the below command to check a timer with clock name CLOCK_REALTIME:

```
$ clocktest -C 0
```

- Use the below command to check a timer with clock name CLOCK_MONOTONIC:

```
$ clocktest -C 1
```

- Use the below command to check a timer with clock name CLOCK_HOST_REALTIME:

```
$ clocktest -C 32
```

2. The interrupts handled by Cobalt : IFC and e1000e interrupts are handled by the Cobalt kernel.

```
$ cat /proc/xenomai/irq
```

NOTE

For e1000e test case, the Linux kernel standard network stack is used instead of rtnet stack.

3. Cobalt IPIPE tracer: The following options are available while configuring the kernel settings:

   a. CONFIG_IPIPE_TRACE_ENABLE (Enable tracing on boot): Defines if the tracer is active by default when booting the system or shall be later enabled via `/proc/ipipe/trace/enable`. Specifically if function tracing is enabled, deferring to switch on the tracer reduces the boot time on low-end systems.

   b. CONFIG_IPIPE_TRACE_MCOUNT (Instrument function entries): Traces each entry of a kernel function. Note that this instrumentation, though it is the most valuable one, has a significant performance impact on low-end systems (~50% larger worst-case latencies on a Pentium-I 133 MHz).

   c. CONFIG_IPIPE_TRACE_IRQSOFF (Trace IRQs-off times): Instruments each disable and re-enable of hardware IRQs. This allows to identify the longest path in a system with IRQs disabled.

   d. CONFIG_IPIPE_TRACE_SHIFT (Depth of trace log): Controls the number of trace points. The I-pipe tracer maintains four ring buffers per CPU of the given capacity in order to switch traces in a lock-less fashion with respect to potentially pending output requests on those buffers. If you run short on memory, try reducing the trace log depth which is set to 16000 trace points by default.

   e. CONFIG_IPIPE_TRACE_VMALLOC (Use vmalloc'ed trace buffer): Instead of reserving static kernel data, the required buffer is allocated via `vmalloc` during boot-up when this option is enabled. This can help to start systems that are low on memory, but it slightly degrades overall performance. Try this option when a traced kernel hangs unexpectedly at boot time.

4. Latency of timer IRQ

```
$ latency -t 2 -T 60
```

NOTE

The location of 'latency' might differ from version to version. Currently it is located in `/usr/bin`.

5. Latency of task in Linux kernel

```
$ latency -t 1 -T 60
```

6. Latency of task in user space

```
$ latency -t 0 -T 60
```

7. Smokey to check feature enabled

```
$ smokey --run
```

8. Thread context switch

```
$ switchtest -T 30
```

9. `xeno-test`: By default, the load command is `dohell 900`, which generates load during 15 minutes.

```
Step #1: Prepare one storage disk and ethernet port connected server, for example:
$ fdisk /dev/sda
$ mkfs.ext2 /dev/sda1
$ mount /dev/sda1 /mnt
$ ifconfig <nw port> <ip addr>

Step #2:
$ cd /usr/xenomai/bin

Step #3:
$ sudo ./xeno-test -l "dohell -s <server ip> -m /mnt"
```

4.3.2 RTnet

RTnet is a protocol stack that runs between the Ethernet layer and the application layer (or IP layer). It aims to provide deterministic communication, by disabling the collision detection CSMA/CD, and preventing buffering packets in the network, through the use of time intervals (time-slots).

RTnet is a software developed to run on Linux kernel with RTAI or Xenomai real-time extension. It exploits the real time kernel extension to ensure the determinism on the communication stack. To accomplish this goal, all the instructions related to this protocol make use of real time kernel functions rather than those of Linux. This binds the latencies to the execution times and latencies of interruptions, which provides deterministic communication.

The following sections describe how to enable the RTnet feature in Xenomai and enable data path acceleration architecture (DPAA) for Xenomai RTnet.

4.3.2.1 Hardware requirements

Following are the hardware requirements for implementing the RTnet protocol in your design:

- For LS1043A, two LS1043ARDB boards (one used as a master and one as a slave board).
- For LS1046A, two LS1046ARDB boards (one used as a master and one as a slave board).
- In case three or more boards are used, a switch is required for connecting all boards into a subnet.
- If you use an e1000e NIC, insert the e1000e NIC into the P4 slot of the LS1043 or LS1046ARDB board.

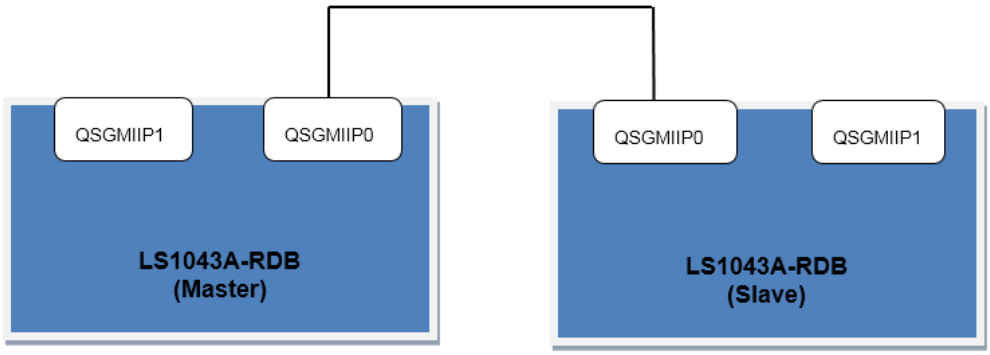


Figure 3. Hardware setup for RTnet (LS1043A as an example)

4.3.2.2 Software requirements

Use the following steps for enabling the RTnet functionality on a Xenomai supported network.

1. Run the command below to configure LS1043ARDB in the `openil` directory:

```
make nxp_ls1043ardb-64b_defconfig
```

   For configuring LS1046ARDB in the `openil` directory, use the command below:

```
make nxp_ls1046ardb-64b_defconfig
```

2. Then, configure the Linux kernel according to the steps listed below.

   For DPAA devices:

   - Disable the Linux DPAA driver using the settings below:

```
$make linux-menuconfig
     Device Drivers --->
        [*] Staging drivers --->
                [ ] Freescale Datapath Queue and Buffer management
```

   - Add the Xenomai RTnet driver and protocol stack using the commands below:

```
$make linux-menuconfig
    [*] Xenomai/cobalt --->
          Drivers --->
                RTnet --->
                    <M> RTnet, TCP/IP socket interface
                    Protocol Stack --->
                        <M> RTmac Layer  --->
                             <  > TDMA discipline for RTmac
                             < M > NoMAC discipline for RTmac
                    Drivers --->
                         <M> FMAN independent mode
```

   For e1000e devices:

   - Disable the Linux e1000e driver using the settings below:

```
$make linux-menuconfig
    Drivers --->
```

```
        [*] Network device support --->
            [*] Ethernet driver support --->
                < > Intel(R) PRO/1000 PCI-Express Gigabit Ethernet support
```

- Add the Xenomai RTnet driver and protocol stack using the commands below:

```
$make linux-menuconfig
  [*] Xenomai/cobalt ---> Drivers --->
                              RTnet --->
                                    <M> RTnet, TCP/IP socket interface Protocol Stack --->
                                        <M> RTmac Layer --->
                                                < > TDMA discipline for RTmac
                                                <M> NoMAC discipline for RTmac Drivers --->
                                                <M> New Intel(R) PRO/1000 PCIe (Gigabit)
```

3. Now, run the `make` command to build all images.
4. After flashing images to the SD card, boot LS1043ARDB or LS1046ARDB from the SD card and enter the Linux prompt.
5. Edit the configuration file, located by default, in the `/etc/rtnet.conf` directory using the settings below:
   a. **DPAA devices**
      - **Master board**
        - RT_DRIVER= "rt_fman_im" - The driver used (currently, it is 'rt_fman_im').
        - IPADDR="192.168.100.101" - IP address of the master board.
        - NETMASK="255.255.255.0" - The other slave board will have the IP 192.168.100.XXX.
        - TDMA_MODE="master"
        - TDMA_SLAVES="192.168.100.102" – If there are two slave boards, this will be "192.168.100.102 192.168.100.103".
      - **Slave board**
        - RT_DRIVER= "rt_fman_im" - The driver used (currently, it is 'rt_fman_im').
        - IPADDR="192.168.100.102" - IP address of the slave board.
        - NETMASK="255.255.255.0" - net mask
        - TDMA_MODE="slave"
        - TDMA_SLAVES="192.168.100.102" – If there are two slave boards, this will be "192.168.100.102 192.168.100.103".
   b. **e1000e devices:**
      - **Master board**
        - RT_DRIVER= "rt_e1000e" - The driver used (currently, it is 'rt_e1000e').
        - IPADDR="192.168.100.101" - IP address of the master board.
        - NETMASK="255.255.255.0" - The other slave board will have the IP 192.168.100.XXX.
        - TDMA_MODE="master"
        - TDMA_SLAVES="192.168.100.102" – If there are two slave boards, this will be "192.168.100.102 192.168.100.103".
      - **Slave board**
        - RT_DRIVER= "rt_e1000e" - The driver used (currently, it is 'rt_e1000e').
        - IPADDR="192.168.100.102" - IP address of the slave board.

- NETMASK="255.255.255.0” - net mask
- TDMA_MODE="slave"
- TDMA_SLAVES="192.168.100.102” – If there are two slave boards, this will be “192.168.100.102 192.168.100.103”.

4.3.2.3 Verifying RTnet

Use the following steps to verify your RTnet connection:

- Step1: Load all modules related with Xenomai RTnet and analyze the configuration file **both** on master and slave sides.

```
$ rtnet start
```

- Use **CTRL+ C**key combination to exit after using the preceding command, if it does not exit on its own.
- Use the below command to display all ethernet ports. Currently, it should display four Ethernet ports (QSGMII Port 0 to Port 3) on master and slave:

```
$ rtifconfig -a
```

- Configure the network on the master side using the commands below:

```
$ rtifconfig rteth0 up 192.208.100.101
$ rtroute solicit 192.208.100.102 dev rteth0
```

- Configure the network on the slave side using the command below:

```
$ rtifconfig rteth0 up 192.208.100.102
```

NOTE

If there are more than one slave boards, you should redo this step using the IP address of the used boards.

- Verify the network connection using the command below:

```
$ rtping 192.208.100.102
```

4.4 IEEE 1588

This section provides an introduction to the IEEE 1588 features of Open IL. It includes a description of the Precision Time Protocol (PTP) device types, Linux PTP stack, quick start guide for implementing PTP based on the IEEE standard 1588 for Linux, known issues and limitations, and long term test results.

4.4.1 Introduction

IEEE Std 1588-2008 (IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems) defines a protocol enabling precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing, and distributed objects.

The 1588 timer module on NXP QorIQ platform provides hardware assist for 1588 compliant time stamping. Together with a software PTP (Precision Time Protocol) stack, it implements precision clock synchronization defined by this standard. Many open source PTP stacks are available with a little transplant effort, such as linuxptp, which are used for this release demo.

4.4.2 PTP device types

There are five basic types of PTP devices, as follows:

- Ordinary clock: A clock that has a single Precision Time Protocol (PTP) port in a domain and maintains the timescale used in the domain. It may serve as a source of time (be a master clock) or may synchronize to another clock (be a slave clock).
- Boundary clock: A clock that has multiple Precision Time Protocol (PTP) ports in a domain and maintains the timescale used in the domain. It may serve as a source of time (be a master clock) or may synchronize to another clock (be a slave clock).
- End-to-end transparent clock: A transparent clock that supports the use of the end-to-end delay measurement mechanism between slave clocks and the master clock.
- Peer-to-peer transparent clock: A transparent clock that, in addition to providing Precision Time Protocol (PTP) event transit time information, also provides corrections for the propagation delay of the link connected to the port receiving the PTP event message. In the presence of peer-to-peer transparent clocks, delay measurements between slave clocks and the master clock are performed using the peer-to-peer delay measurement mechanism.
- Management node: A device that configures and monitors clocks.

NOTE

Transparent clock, is a device that measures the time taken for a Precision Time Protocol (PTP) event message to transit the device and provides this information to clocks receiving this PTP event message.

4.4.3 Linux PTP stack

The Linux PTP stack software is an implementation of the Precision Time Protocol (PTP) based on the IEEE standard 1588 for Linux. Its dual design goals are:

- To provide a robust implementation of the standard.
- To use the most relevant and modern Application Programming Interfaces (API) offered by the Linux kernel.

Supporting legacy APIs and other platforms is not an objective of this software. Following are the main features of the Linux PTP stack:

- Supports hardware and software time stamping via the Linux SO_TIMESTAMPING socket option.
- Supports the Linux PTP Hardware Clock (PHC) subsystem by using the `clock_gettime` family of calls, including the new `clock_adjtimex` system call.
- Implements Boundary Clock (BC) and Ordinary Clock (OC).
- Transport over UDP/IPv4, UDP/IPv6, and raw Ethernet (Layer 2).
- Supports IEEE 802.1AS-2011 in the role of end station.
- Modular design allows painless addition of new transports and clock servo algorithms.

4.4.4 Quick start guide for setting up IEEE standard 1588 demonstration

This quick start guide explains the procedure to set up demos of IEEE 1588, including master-slave synchronization, boundary clock synchronization, and transparent clock synchronization.

1. **Hardware requirement**
   - Two boards for basic master-slave synchronization
   - Three or more boards for BC synchronization
   - Three or more boards for TC synchronization (One must be LS1021ATSN board)

2. **Software requirement**
   - Linux BSP of industry solution release
   - PTP software stack
3. **Ethernet interfaces connection for master-slave synchronization**

   Connect two Ethernet interfaces between two boards in a back-to-back manner. Then, one board works as master and the other works as a slave when they synchronize. Both the master and the slave work as Ordinary Clocks (OCs).
4. **Ethernet interfaces connection for BC synchronization**

   At least three boards are required for BC synchronization. When three boards are used for BC synchronization, assuming board A works as boundary clock (BC) with two PTP ports, board B and board C work as OCs.

Table 16. Connecting Ethernet interfaces for boundary clocks (BC) synchronization

Board	Clock type	Interfaces used
A	BC	Interface 1, Interface 2.
B	OC	Interface 1
C	OC	Interface 1

5. Connect board A interface 1 to board B interface 1 in back-to-back manner.
6. Connect board A interface 2 to board C interface 1 in back-to-back manner. For example, LS1021ATSN BC synchronization connection is shown in the following figure.

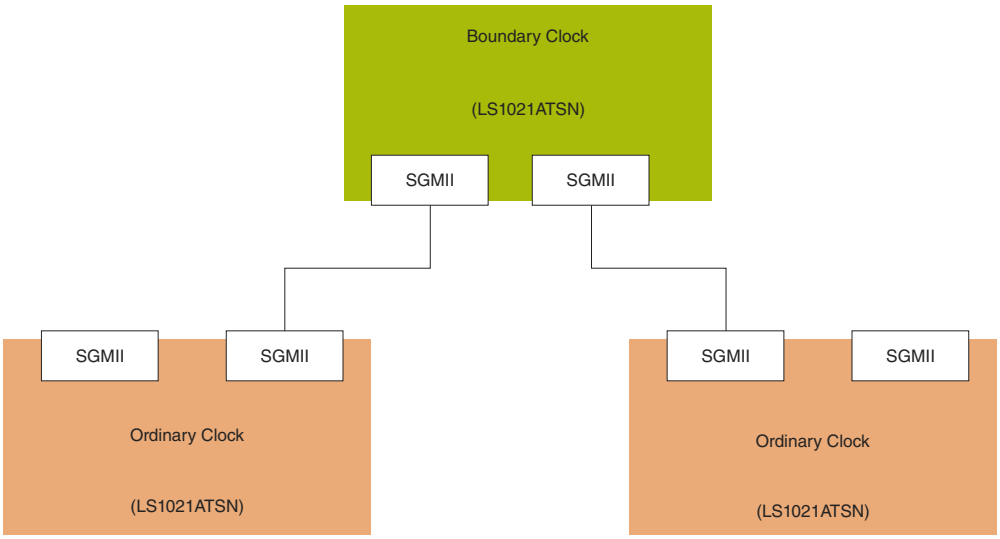


Figure 4. LS1021ATSN BC synchronization

7. **Ethernet interfaces connection for transparent clock (TC) synchronization**

   At least three boards are required for TC synchronization. One must be LS1021ATSN board, which is needed as a transparent clock since there is a SJA1105 switch on it. When three boards are used for TC synchronization, assuming board A (LS1021ATSN) works as TC with two PTP ports, board B and board C work as OCs.

NOTE

i.MX6Q SabreSD supports only the master-slave mode.

Table 17. Connecting Ethernet interfaces for TC (transparent clock)

Board	Clock Type	Interfaces used
A (LS1021ATSN)	TC	Interface 1, Interface 2. (These are two ports of SJA1105 switch.)
B	OC	Interface 1
C	OC	Interface 1

- Connect board A interface 1 to board B interface 1 in a back-to-back manner.
- Connect board A interface 2 to board C interface 1 in a back-to-back manner. For example, LS1021ATSN TC synchronization connection is shown in the following figure.

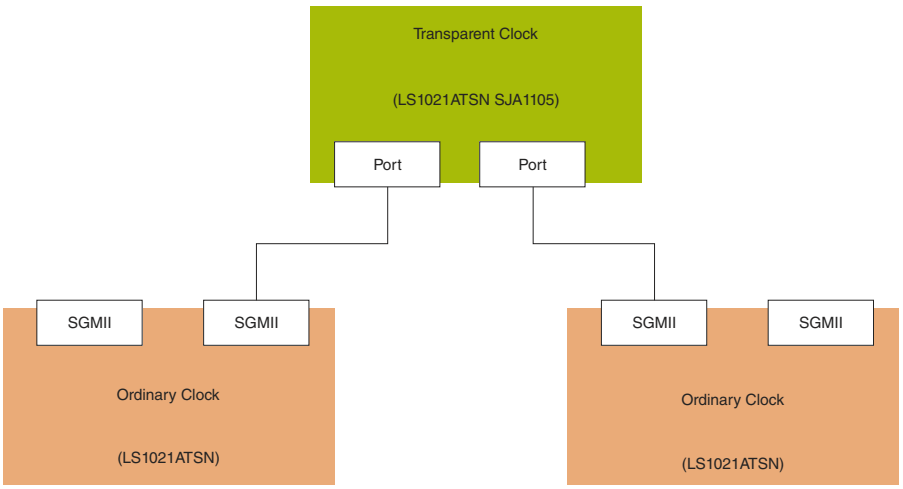


Figure 5. LS1021ATSN TC synchronization

8. **PTP stack startup**

   Before starting up the kernel to run PTP stack, make sure there is no MAC address conflict in the network. Different MAC addresses should be set for each MAC on each board in U-Boot. For example,

   Board A:

```
=> setenv ethaddr 00:04:9f:ef:00:00
=> setenv eth1addr 00:04:9f:ef:01:01
=> setenv eth2addr 00:04:9f:ef:02:02
```

   Board B:

```
=> setenv ethaddr 00:04:9f:ef:03:03
=> setenv eth1addr 00:04:9f:ef:04:04
=> setenv eth2addr 00:04:9f:ef:05:05
```

Board C:

```
=> setenv ethaddr 00:04:9f:ef:06:06
=> setenv eth1addr 00:04:9f:ef:07:07
=> setenv eth2addr 00:04:9f:ef:08:08
```

Linux PTP stack supports both OC and BC. It is included in the SD card images of LS1021ATSN, LS1043ARDB, LS1046ARDB, and i.MX6Q SabreSD, built using buildroot.

9. **Basic master-slave synchronization**

   For basic master-slave synchronization, use the below command. It can be observed that the slave synchronizes with the master with time.

   - **For LS platforms:**

     ```
     $ ptp4l -i eth0 -p /dev/ptp0 -f /etc/ptp4l_default.cfg –m
     ```

   - **For i.MX platforms**:

     First create ptp config file as follow for both board A and B:

     ```
     cat ptp.cfg
     [global]
     #
     # Run time options
     #
     logAnnounceInterval -4
     logSyncInterval -4
     logMinDelayReqInterval -4
     logMinPdelayReqInterval -4
     tx_timestamp_timeout 10
     ```

     - **Board A**

       ```
       sysctl -w  net.ipv4.igmp_max_memberships=20
       ifconfig eth0 up 192.168.0.100
       ptp4l -f ./ptp.cfg -A -4 -H -m -i eth0
       ```

     - **Board B**

       ```
       sysctl -w  net.ipv4.igmp_max_memberships=20
       ifconfig eth0 up 192.168.0.101
       ptp4l -f ./ptp.cfg -A -4 -H -m -i eth0
       ```

10. **BC synchronization**

    For BC synchronization, run OC using the below command. It can be observed that the slave synchronizes with the master with time.

    ```
    $ ptp4l -i eth0 -p /dev/ptp0 -f /etc/ptp4l_default.cfg -m
    ```

    If the board is used as BC with several PTP ports, the ‘-i ’ argument could point more interfaces. For running BC with more than one interfaces, use the below command:

    ```
    $ ptp4l -i eth0 -i eth1 -p /dev/ptp0 -f /etc/ptp4l_default.cfg –m
    ```

11. **TC synchronization**

For TC synchronization, set the two-step end-to end transparent clock configuration for SJA1105 on TC (LS1021ATSN). Free running PTP clock is used for TC because the residence time is very short (about 2 ~ 3 µs as per test results). Even if synchronization is implemented for TC, the improvement for residence time accuracy is still very small and can be ignored.

```
$ sja1105-ptp-free-tc.sh
```

Run OC using the below command:

```
 $ ptp4l -i eth0 -p /dev/ptp0 -2 -m
```

It can be observed that slave synchronizes its time with the master clock. If you use the '-l 7' argument to enable debug message for slave, the correction field value of Sync and Delay_resp messages are displayed, which are the residence time of Sync and Delay_req messages.

NOTE

- For all the three cases mentioned above, the master clock could be selected by using the software BMC (Best Master Clock) algorithm.
- The interface name and PTP device name in commands should be changed accordingly.

4.4.5 Known issues and limitations

1. For LS1021ATSN, the Linux PTP stack only supports LS1021A Ethernet interfaces. It cannot be used for SJA1105 switch Ethernet interfaces.
2. Packet loss issue could be observed on LS1021ATSN SGMII interfaces connected in back-to-back manner. The root cause is that the PHY supports IEEE 802.11az EEE mode, by default. The low speed traffic makes it switch to low power mode, which affects 1588 synchronization performance greatly.

   Use the below workaround to disable this feature.

   ```
   $ ifconfig eth0 up
   $ ethtool --set-eee eth0 advertise 0
   $ ifconfig eth0 down
   $ ifconfig eth0 up
   ```

3. The ptp4l stack may report a timeout for getting the tx timestamp, but this rarely appears. This is not a bug. The stack tries to get the tx timestamp after sending a message, but cannot get it if the driver has not completed tx timestamp processing, in time. Just increasing the tx_timestamp_timeout parameter and re-running the stack will resolve this problem.

   ```
   ptp4l[574.149]: timed out while polling for tx timestamp
   ptp4l[574.152]: increasing tx_timestamp_timeout may correct this issue, but it is likely caused by a driver bug
   ```

4.4.6 Long term test results for Linux PTP

This section describes the long term test results for Linux PTP stack implementation.

Linux PTP

Connection: back-to-back master to slave

Configuration: Sync internal is -3

Test boards: two LS1021ATSN boards, one as master and another one as slave

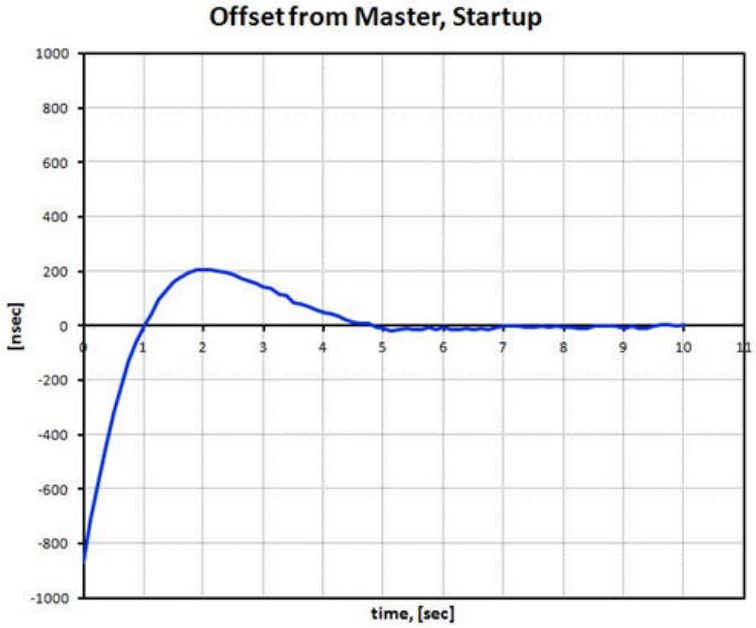


Figure 6. Offset from master in start up state

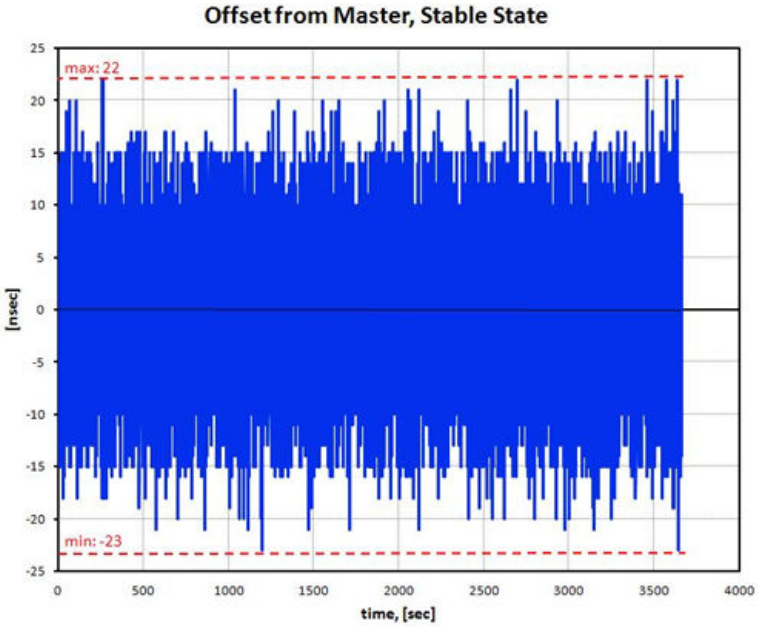


Figure 7. Offset from master in stable state

4.5 OP-TEE

This section explains how to run Open Portable Trusted Execution Environment (OP-TEE) on ARM® based NXP platforms, such as LS1021A-TSN and LS1021A-IoT platforms. OP-TEE started as collaboration between ST Microelectronics and Linaro. Later, it was made available to the open source community. It contains the complete stack from normal world client APIs (`optee_client`), the Linux kernel TEE driver (`optee_linuxdriver`), and the Trusted OS and the secure monitor (`optee_os`).

4.5.1 Introduction

This section describes the operating environment, tools and, dependencies necessary for deploying OP-TEE. It describes the installation based on the design and setup of one specific environment. Thereafter, users need to adapt the setup and deployment for their specific environment and requirements.

It includes the following:

- Getting OP-TEE and relevant test program
- Compiling the image
- Prerequisites of integrating TEE binary image into the final images.
- Installation and usage steps for the TEE application and output obtained on the LS1021A platform.

The TEE used for this demo is Open Portable Trusted Execution Environment (OP-TEE).

This release supports the following features:

- Supports the LS1021A-TSN and LS1021A-IOT platforms
- Secure boot by SD boot
- TrustZone Controller enabled
- U-boot: v2016.09.
- Linux Kernel v4.1 with OP-TEE drivers backported from mainline kernel v4.11
- OP-TEE OS: v2.4.0
- OP-TEE Client: v2.4.0
- OP-TEE Test: v2.4.0.

NOTE

For LS1021AIOT, the `nxp_ls1021aiot_optee_defconfig` configuration file does not support secure boot, it just includes OP-TEE.

4.5.2 Deployment architecture

The following figure shows the deployment architecture of OP-TEE on ARM TrustZone enabled SoCs.

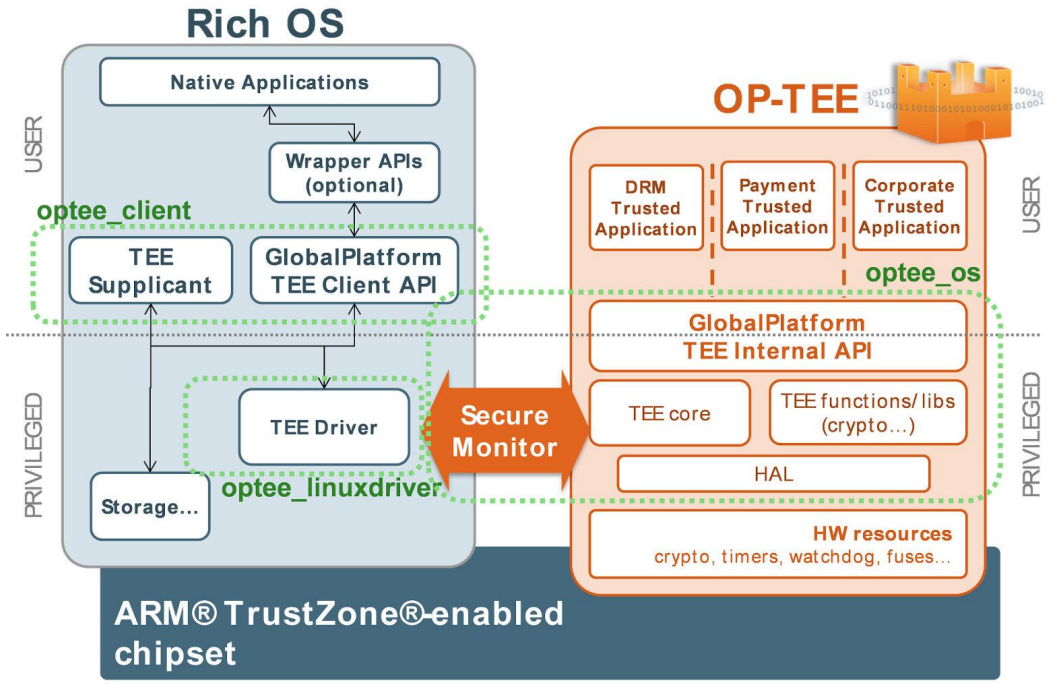


Figure 8. Architecture of OP-TEE on an ARM TrustZone enabled SoC

4.5.3 DDR memory map

The following figure shows the DDR memory map for LS1021A-TSN platform with OP-TEE implementation.

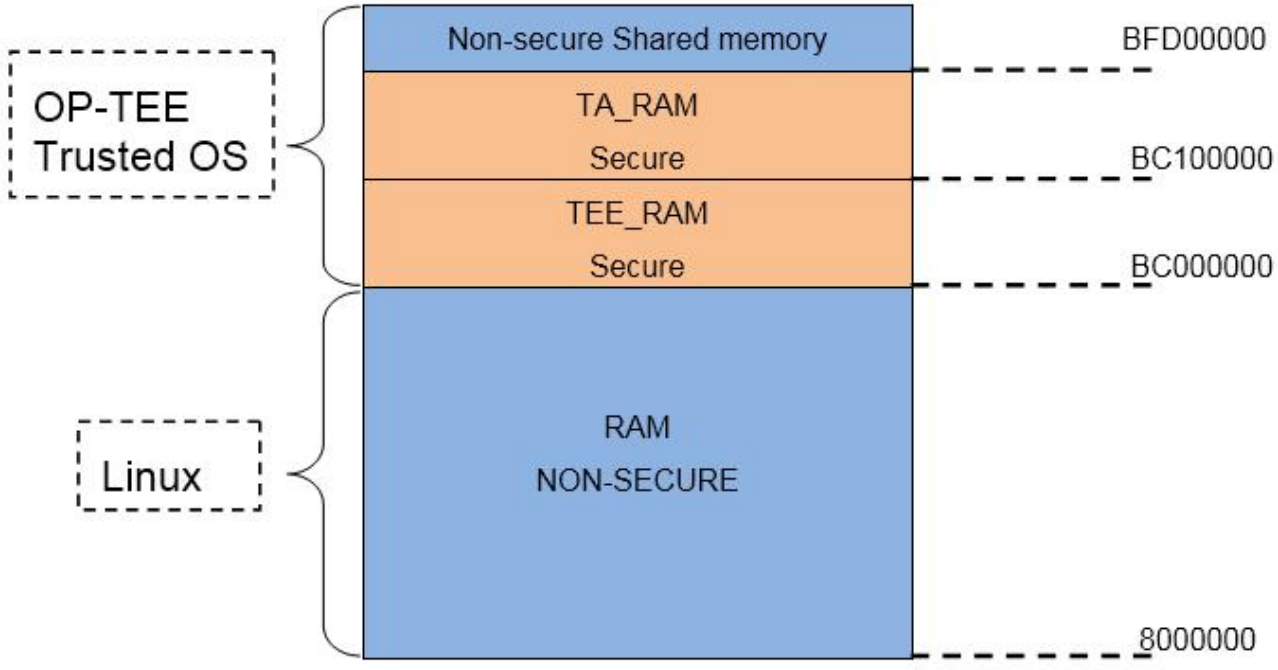


Figure 9. DDR memory map

4.5.4 Configuring OP-TEE on LS1021A-TSN platform

Use the following commands to build the images with the OP-TEE feature on the LS1021A-TSN platform.

```
$ cd openil
$ make clean
$ make nxp_ls1021atsn_optee-sb_defconfig
$ make
#or make with a log
$ make 2>&1 | tee build.log
```

NOTE

The host Linux machine must have the following libraries:

- `libmagickwand-dev` for APT on Debian/Ubuntu.
- `ImageMagick-devel` for Yum on CentOS.

The `nxp_ls1021atsn_optee-sb_defconfig` configuration file includes some default configurations for secure boot and OP-TEE. These are listed below:

1. `ls1021atsn_sdcard_SECURE_BOOT_TEE` U-Boot configuration.
2. `kernel CONFIG_OPTEE` configuration.
3. OP-TEE OS, client, and test applications.
4. `CST tool` to create secure boot keys and headers.

The CST tool can support two special functions, which are:

1. **Using custom srk.pri and srk.pub files to maintain the consistent keys.** For this feature, move the custom srk.pri and srk.pub files into the directory named `board/nxp/ls1021atsn/`. Then, the CST tool creates all the keys and header files for secure boot based on the two files, each time. In addition, after running `gen_keys 1024` to get the `srk.pri` and `srk.pub files` at the first instance, if there are no custom files in `board/nxp/ls1021atsn/`, the CST tool always uses the existing srk.pri and srk.pub, until the two files are deleted.

2. **Enabling/disabling the core hold-off switch for the secure boot, by using the `make menuconfig` command**.

   This can be done by using the following command:

   ```
   Host utilities  --->
   [*]host cst tool
   *** core hold-off ***
      [*] secure boot core hold-off
   ```

   After the correct building, the final SD card image named `sdcard.img` can be located at `output/images`. The keys for secure boot that should be programmed into the silicon can be located in the file `output/images/srk.txt`.

4.5.5 Running OP-TEE on LS1021A-TSN platform

This section provides the commands for running OP-TEE on the LS1021A-TSN platform. It includes commands for secure boot, executing OP-TEE daemon, and executing OP-TEE test cases.

4.5.5.1 Running secure boot

OP-TEE must run together with secure boot in order to protect all images to avoid being attacked. For details about secure boot, refer to the section, *Secure Boot* in the Chapter, *Boot Loaders* in the online LSDK document: https://freescalereach01.sdlproducts.com/LiveContent/web/pub.xql?c=t&action=home&pub=QorIQ_SDK&lang=en-US

Refer to the following useful CCS commands for secure boot:

```
#Connect to CCS and configure Config Chain
delete all
config cc cwtap:<ip address of cwtap> show cc
ccs::config_server 0 10000
ccs::config_chain {ls1020a dap sap2} display ccs::get_config_chain

#Check Initial SNVS State and Value in SCRATCH Registers
ccs::display_mem <dap chain pos> 0x1e90014 4 0 4
ccs::display_mem <dap chain pos> 0x1ee0200 4 0 4

#Write the SRK Hash Value in Mirror Registers
ccs::write_mem <dap chain pos> 0x1e80254 4 0 <SRKH1>
ccs::write_mem <dap chain pos> 0x1e80258 4 0 <SRKH2>
ccs::write_mem <dap chain pos> 0x1e8025c 4 0 <SRKH3>
ccs::write_mem <dap chain pos> 0x1e80260 4 0 <SRKH4>
ccs::write_mem <dap chain pos> 0x1e80264 4 0 <SRKH5>
ccs::write_mem <dap chain pos> 0x1e80268 4 0 <SRKH6>
ccs::write_mem <dap chain pos> 0x1e8026c 4 0 <SRKH7>
ccs::write_mem vdap chain pos> 0x1e80270 4 0 <SRKH8>

#Get the Core Out of Boot Hold-Off
ccs::write_mem <dap chain pos> 0x1ee00e4 4 0 0x00000001
```

If the image verification passes, the board boot up starts in the secure mode.

4.5.5.2 Executing Op-tee Daemon

Run OPTee client daemon using the command below:

```
tee-supplicant /dev/teepriv0 &
```

4.5.5.3 Executing OP-Tee test cases

OP-Tee test cases can be run using the steps listed below.

1. Run xtest binary in Linux console:

   ```
   xtest
   ```

2. Then you should get a log similar to the following as a test result:

   ```
   Run test suite with level=0
   TEE test application started with device [(null)]
   ########################################################
   #
   # regression
   #
   ########################################################
   …
   24003 subtests of which 0 failed
   76 test cases of which 0 failed
   0 test case was skipped
   TEE test application done!
   ```

4.6 SELinux

SELinux is a security enhancement to Linux that allows users and administrators better access control.

Access can be constrained on variables so as to enable specific users and applications to access specific resources. These resources may take the form of files. Standard Linux access controls, such as file modes (-rwxr-xr-x) are modifiable by the user and the applications which the user runs. Conversely, SELinux access controls are determined by a policy loaded on the system, which are not changed by careless users or misbehaving applications.

SELinux also adds finer granularity to access controls. Instead of only being able to specify who can read, write or execute a file, for example, SELinux lets you specify who can unlink, append only, move a file, and so on. SELinux allows you to specify access to many resources other than files as well, such as network resources and interprocess communication (IPC).

More information can be found at official Security Enhanced Linux (SELinux) project page: https://selinuxproject.org.

4.6.1 Running SELinux demo

This section describes the procedure for running the SELinux demo on NXP's LS1043ARDB-64bit and LS1046ARDB-64bit platforms.

4.6.1.1 Obtaining the image for SELinux

The SELinux can run on NXP LS1043ARDB-64bit and LS1046ARDB-64bit with Ubuntu file system.

Use the below commands for building these two platforms for the SELinux demo:

```
$ cd openil
$ make clean
$ make nxp_ls1043ardb-64b_ubuntu_defconfig
# or ls1046ardb-64b platform
$ make nxp_ls1046ardb-64b_ubuntu_defconfig
$ make
# or make with a log
$ make 2>&1 | tee build.log
```

4.6.1.2 Installing basic packages

Install the following basic packages before running the SELlinux demo:

1. Basic packages:
   - $ apt-get update
   - $ apt-get install dpkg
   - $ apt-get install vim
   - $ apt-get install wget
   - $ apt-get install bzip2
   - $ apt-get install patch
   - $ apt-get install bison
   - $ apt-get install flex
   - $ apt-get install xz-utils
   - $ apt-get install selinux-utils
   - $ apt-get install policycoreutils

- $ apt-get install auditd
- $ apt-get install ssh
- $ apt-get install apache2
- $ apt-get install selinux-basics
- $ apt-get install selinux-policy-default

2. Install Grub Common from source code:

```
$ wget http://archive.ubuntu.com/ubuntu/pool/main/g/grub2/grub2_2.02~beta2.orig.tar.xz
$ tar -xvf grub2_2.02~beta2.orig.tar.xz$ cd grub-2.02~beta2/
$ ./configure
$ make
$ make install
```

NOTE

It would take considerable time to run the `make` and `make install` commands.

3. Install SELinux from source code:

```
$ wget http://archive.ubuntu.com/ubuntu/pool/universe/s/selinux/selinux_0.11.tar.gz
$ tar -xvf selinux_0.11.tar.gz
$ cd selinux-0.10/
$ make
$ make install
```

4. Install SELinux policy from source code:

```
$ wget http://archive.ubuntu.com/ubuntu/pool/universe/r/refpolicy-ubuntu/refpolicy-
ubuntu_0.2.20091117.orig.tar.bz2
$ wget http://archive.ubuntu.com/ubuntu/pool/universe/r/refpolicy-ubuntu/refpolicy-
ubuntu_0.2.20091117-0ubuntu2.debian.tar.gz
$ tar jxvf refpolicy-ubuntu_0.2.20091117.orig.tar.bz2
$ tar zxvf refpolicy-ubuntu_0.2.20091117-0ubuntu2.debian.tar.gz
$ cd refpolicy
$ cp -r ../debian/patches/ ./
patch -p1 < patches/bashisms.patch
patch -p1 < patches/*.patch
patch -p1 < patches/conf.patch
patch -p1 < patches/users.patch
patch -p1 < patches/xserver.patch
patch -p1 < patches/sysnetwork.patch
patch -p1 < patches/cups.patch
patch -p1 < patches/ssh.patch
patch -p1 < patches/hal.patch
patch -p1 < patches/dbus.patch
patch -p1 < patches/bluetooth.patch
patch -p1 < patches/avahi.patch
patch -p1 < patches/networkmanager.patch
patch -p1 < patches/consolekit.patch
patch -p1 < patches/usermanage.patch
patch -p1 < patches/cron.patch
patch -p1 < patches/corecommands.patch
patch -p1 < patches/userdomain.patch
patch -p1 < patches/fstools.patch
patch -p1 < patches/kernel.patch
patch -p1 < patches/locallogin.patch
patch -p1 < patches/unconfined.patch
```

```
patch -p1 < patches/libraries.patch
patch -p1 < patches/init.patch
patch -p1 < patches/mount.patch
patch -p1 < patches/udev.patch
patch -p1 < patches/devtmpfs.patch
patch -p1 < patches/rtkit.patch
patch -p1 < patches/devkit.patch
patch -p1 < patches/gnome.patch
patch -p1 < patches/apt.patch
patch -p1 < patches/policykit.patch
patch -p1 < patches/modemmanager.patch
patch -p1 < patches/fix-ftbfs.patch
$ make conf
$ make policy
$ make install
```

4.6.1.3 Basic setup

Perform the following basic steps before running the SELlinux demo.

1. Map root to sysadm_u, modify the mapping of root and selinux user:

   ```
   $ semanage login -m -s sysadm_u root
   ```

   Logout and login again. Check root's SELinux login user:

   ```
   $ id -Z
   sysadm_u:sysadm_r:sysadm_t:s0
   ```

2. Map linux user to a selinux user named user_u:

   ```
   $ semanage login -m -s user_u __default__
   ```

   Check all the selinux users logged in:

   ```
   $ semanage login -l

   Login Name           SELinux User         MLS/MCS Range        Service

   __default__           user_u               s0                     *
   root                  sysadm_u             s0                     *
   system_u              system_u             s0-s0:c0.c1023         *
   ```

3. Label the system. Modify the SELinux config file with `SELINUXTYPE=default` using the command below:

   ```
   $ vim /etc/selinux/config
   ```

   Restore the type of files in /root:

   ```
   $ semanage fcontext -a -t home_root_t '/root(/.*)?'
   ```

   Restore the system using the command below:

   ```
   $ restorecon -R /
   $ reboot
   ```

4. Check ssh server after the kernel boots up:

```
$ systemctl status ssh
ssh.service - OpenBSD Secure Shell server
Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled) Active: active
(running) since 2017-05-09 07:23:56 CST; 1 weeks 6 days ago
Main PID: 908 (sshd)
CGroup: /system.slice/ssh.service
└─908 /usr/sbin/sshd -D
```

   If checking the ssh server status fails, restart the ssh server using the command below:

```
$ systemctl restart ssh
```

5. Check the http server:

```
$ systemctl status apache2
 └─apache2.service - LSB: Apache2 web server
Loaded: loaded (/etc/init.d/apache2; bad; vendor preset: enabled) Drop-In: /lib/systemd/system/
apache2.service.d
└─apache2-systemd.conf
Active: active (running) since Thu 2016-02-11 16:30:39 UTC; 2min 3s ago Docs: man:systemd-sysv-
generator(8)
Process: 3975 ExecStart=/etc/init.d/apache2 start (code=exited, status=0/SUCCE CGroup: /
system.slice/apache2.service
├─3990 /usr/sbin/apache2 -k start
├─3993 /usr/sbin/apache2 -k start
└─3994 /usr/sbin/apache2 -k start
```

   If checking the apache2 status fails, restart apache2 service:

```
$ systemctl restart apache2
```

6. Add the user test1: Add a linux user named test1. Specify password for test1 and other configurations can be defaultMap root to sysadm_u.

```
$ adduser test1
Adding user `test1' ...
Adding new group `test1' (1001) ...
Adding new user `test1' (1001) with group `test1' ... Creating home directory `/home/test1' ...
Copying files from `/etc/skel' ... Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully Changing the user information for test1
Enter the new value, or press ENTER for the default Full Name []:
Room Number []: Work Phone []: Home Phone []: Other []:
Is the information correct? [Y/n] y
```

4.6.1.4 Demo 1: local access control

This demo shows how SELinux protects a local file. The process cannot access local files if it is unauthorized.

Example 1: Denying a process from reading a wrong file type
In this example, a vi process created by user with uid: test1, acts as a subject to access a common file, which has a DAC permission of 777.

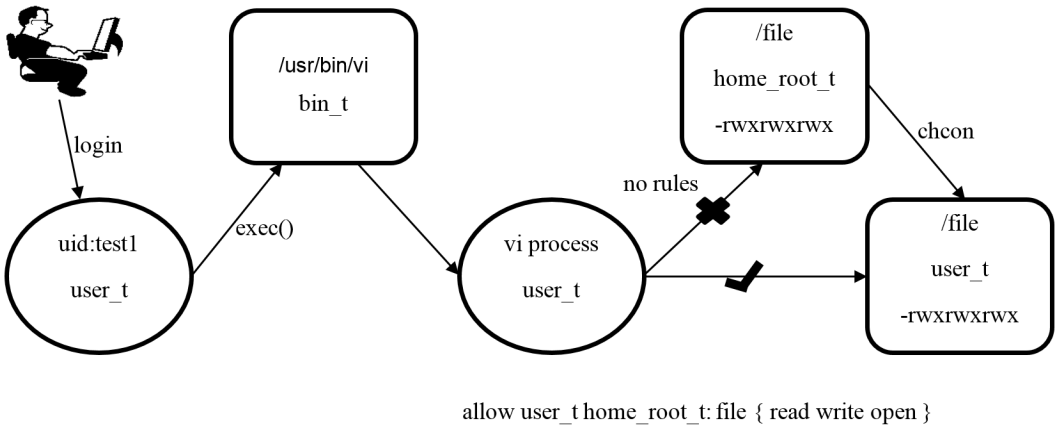


Figure 10. Allowing local file access control

1. root: create a test file:

```
$ echo “file created in root home” > /root/file
$ chmod 777 /root/file
$ mv /root/file /
$ ls -Z /file
sysadm_u:object_r:home_root_t:s0 /file
```

2. root: enable SELinux:

```
$ setenforce 1
$ getenforce 0
Enforcing
```

3. User test1: logs in and visits the file. User test1 logs in the system via ssh and checks id info:

```
$ id -Z
user_u:user_r:user_t:s0
```

User test1 visits the file using the vi command.

```
$ vi /file
```

SELinux denies access to the file, even though the file is 777.

```
"/file" [Permission Denied]
```

Figure 11. The VI command log

Because there is no allowed rule such as the following

```
allow user_t home_root_t: file {write append}
```

4. root: change the type of file

```
$ setenforce 0
$ chcon -t user_t /file
$ setenforce 1
```

5. User test1: visits the file of correct type, and his request is approved. The user test1 visits the file again and succeeds.

```
$ vi /file
```

6. root: Refer to the audit log: `/var/log/audit/audit.log` with commands `audit2why` and `audit2allow`.

```
$ audit2why -a
```

There is an AVC information about access denied and a reasonable root cause as shown in the below figure.

```
type=AVC msg=audit(1455223344.656:204): avc:  denied  { write } for  pid=3263 c
omm="vi" name="file" dev="mmcblk0p3" ino=146470 scontext=user_u:user_r:user_t:s
0 tcontext=sysadm_u:object_r:home_root_t:s0 tclass=file permissive=1
        Was caused by:
                Missing type enforcement (TE) allow rule.

                You can use audit2allow to generate a loadable module to allow
this access.
```

Figure 12. Audit log for vi

```
$ audit2allow -a
```

This command suggests the rules that can approve the access.

```
#============= user_t ==============

#!!!! The source type 'user_t' can write to a 'file' of the following types:
# wireshark_home_t, telepathy_logger_data_home_t, screen_home_t, screen_var_run
_t, xdm_tmp_t, mplayer_tmpfs_t, gconf_tmp_t, rssh_rw_t, httpd_user_script_exec_
t, evolution_home_t, tvtime_home_t, tvtime_tmp_t, httpd_user_content_t, irc_hom
e_t, telepathy_mission_control_data_home_t, pulseaudio_tmp_t, spamassassin_tmp_
t, git_user_content_t, pulseaudio_home_t, telepathy_tmp_content, mozilla_plugin
_tmp_t, rssh_ro_t, wireshark_tmpfs_t, razor_home_t, gift_home_t, user_home_t, p
pp_home_t, xauth_home_t, uml_tmpfs_t, ssh_home_t, pyzor_tmp_t, session_dbusd_tm
p_t, httpd_user_rw_content_t, hadoop_home_t, zookeeper_tmp_t, java_tmpfs_t, use
r_fonts_t, games_tmpfs_t, uml_ro_t, mail_home_rw_t, vmware_conf_t, gpg_agent_tm
p_t, spamd_home_t, krb5_home_t, gnome_home_t, telepathy_sunshine_home_t, telepa
thy_logger_cache_home_t, uml_tmp_t, vmware_file_t, irc_tmp_t, pulseaudio_tmpfsf
ile, user_home_t, gnome_keyring_home_t, evolution_exchange_tmpfs_t, iceauth_hom
e_t, razor_tmp_t, telepathy_mission_control_cache_home_t, cgroup_t, bluetooth_h
elper_tmp_t, pulseaudio_tmpfs_t, wireshark_tmp_t, mail_spool_t, alsa_home_t, th
underbird_home_t, evolution_tmpfs_t, vmware_tmpfs_t, mozilla_tmpfs_t, telepathy
_mission_control_home_t, irc_log_home_t, gift_tmpfs_t, screen_tmp_t, vmware_tmp
_t, mail_home_t, mozilla_tmp_t, mplayer_home_t, pulseaudio_home_t, httpd_user_h
taccess_t, usbfs_t, mozilla_plugin_home_t, telepathy_gabble_cache_home_t, spamc
_tmp_t, evolution_webcal_tmpfs_t, user_mail_tmp_t, spamassassin_home_t, evoluti
on_alarm_tmpfs_t, hadoop_tmp_t, security_t, mysqld_home_t, oidentd_home_t, tele
pathy_cache_home_t, uml_rw_t, user_fonts_config_t, bluetooth_helper_tmpfs_t, mo
zilla_plugin_tmpfs_t, games_tmp_t, user_tmp_t, mozilla_home_t, nfsd_rw_t, uml_e
xec_t, java_tmp_t, mpd_user_data_t, telepathy_data_home_t, tvtime_tmpfs_t, http
d_user_ra_content_t, pyzor_home_t, user_tmpfs_t, user_fonts_cache_t, gpg_secret
_t, anon_inodefs_t, gconf_home_t, xserver_tmpfs_t, session_dbusd_home_t

allow user_t home_root_t:file { read write open };
```

Figure 13. Audit suggestion for Vi

Example 2: Denying a root user from changing SELinux running mode

In this example, the root user is restricted to have no permission to change the SELinux running mode when SELinux is enforced.

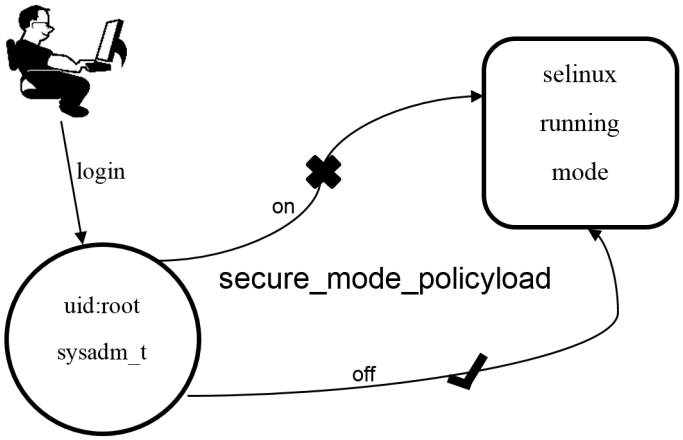


Figure 14. Restricting root permissions

1. Root: Turn on and then turn off Selinux

   Booleans are shortcuts for the user to modify the SELinux policy dynamically. The policy,`secure_mode_policyload` is one of these policies, which can deny a root user from changing SELinux running mode. By default, it is Off.

   ```
   $ getsebool secure_mode_policyload
   secure_mode_policyload --> off
   ```

   Root can turn on SELinux:

   ```
   $ setenforce 1
   ```

   Root can then turn off SELinux:

   ```
   $ setenforce 0
   ```

2. root: `enable secure_mode_policyload`

   Now the SELinux is permissive. Run the `setsebool` command to enable `secure_mode_policyload`:

   ```
   $ setsebool secure_mode_policyload on
   ```

   Check the status of `secure_mode_policyload` again:

   ```
   $ getsebool secure_mode_policyload
   secure_mode_policyload --> on
   ```

3. Root: Try to turn on and turn off SELinux.

   Root can still turn on SELinux:

   ```
   $ setenforce 1
   ```

   Root tries to turn off SELinux but gets permission denied:

   ```
   $ setenforce 0
     setenforce:       setenforce() failed
   ```

If root user tries to disable `secure_mode_policyload`, it fails too:

```
$ setsebool secure_mode_policyload off
Segmentation fault
```

Now there is no superuser in the system even if you are the root user.

Reboot the system. Booting with kernel option `enforcing=0` can make the system running in permissive mode. In this way, you can proceed with use cases similar to the ones described above.

4.6.1.5 Demo 2: enabling remote access control

This demo shows how SELinux can also be used to provide website visiting permissions. A web client cannot access website files remotely if it is not authorized.

Example 1: Denying an HTTP client from visiting a private website

Use the following commands for running this sample demo:

1. root: Copy index.html to /root

```
$ cp /var/www/html/index.html /root
```

2. root: Move index.html to apache2

```
$ mv /root/index.html /var/www/html/index.html
```

3. root: turn on SELinux and `wget` website

```
$ setenforce 1
$ wget localhost
--2016-02-11 16:41:08--  http://localhost/
Resolving localhost [  795.609868] systemd-journald[1983]: recvmsg() failed: Permission denied
(localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:80... connected.
HTTP request sent, awaiting response... 403 Forbidden
2016-02-11 16:41:08 ERROR 403: Forbidden.
```

Now `wget`, as a http client, fails to visit apache2 home page.

4. root: check type of index.html.

```
$ ls -Z /var/www/html/index.html
sysadm_u:object_r:home_root_t:SystemLow /var/www/html/index.html
```

The index.html has a type of home_root_t which cannot be access by the http client with type httpd_t.

5. root: restore index.html to a right type.

```
$ setenforce 0
$ restorecon /var/www/html/index.html
$ ls -Z /var/www/html/index.html
sysadm_u:object_r:httpd_sys_content_t:SystemLow /var/www/html/index.html
```

The index.html now contains the httpd_sys_content_t and can be access by httpd_t.

6. root: turn on SELinux and visit again.

```
$ setenforce 1
$ wget localhost
```

```
Resolving localhost (localhost)... ::1, 127.0.0.1
Connecting to localhost (localhost)|::1|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 11321 (11K) [text/html]
Saving to: 'index.html.1'
index.html.1        100%[===================>]  11.06K  --.-KB/s    in 0s
2016-02-11 16:57:16 (148 MB/s) - 'index.html.1' saved [11321/11321]
```

Example 2 Denying ssh client from remote login with root

The following figure shows how to deny ssh remote login permission for a root user.

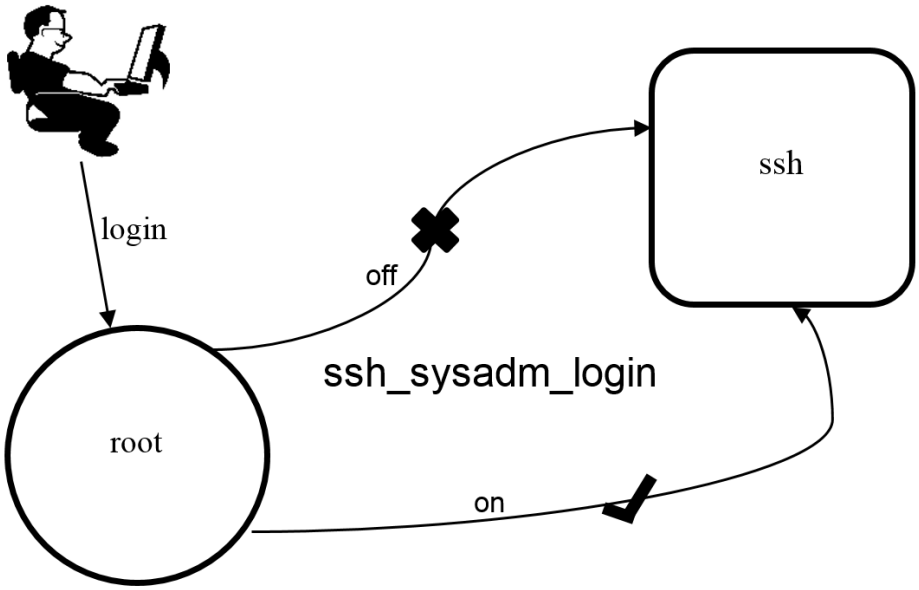


Figure 15. ssh remote permission

1. root: config sshd to permitrootlogin

```
$ setenforce 0
$ vi /etc/ssh/sshd_config
```

   Find “PermitRootLogin prohibit-password” and change it to “PermitRootLogin yes”

2. root: restart ssh server

```
$ /etc/init.d/sshd restart
```

   Now root should be allowed to access the system from remote side with ssh.

3. root: turn on SELinux and ssh.

```
$ setenforce 1
$ ssh root@localhost
/bin/bash: Permission denied
Connection to localhost closed.
```

   Even though sshd_config file has permitted root login but still fails in ssh.

4. root: turn on ssh login boolean

Check that the following settings are configured:

```
$ getsebool -a | grep ssh
allow_ssh_keysign --> off
fenced_can_ssh --> off
sftpd_write_ssh_home --> off
ssh_sysadm_login --> off
ssh_use_gpg_agent --> off
```

There is a boolean named `ssh_sysadm_login`. This denies a root user from ssh login. Turn on it.

```
$ setenforce 0
$ setsebool ssh_sysadm_login on
```

5. root: enforcing and ssh again.

```
$ setenforce 1
$ ssh root@localhost
```

Now root user can ssh successfully.

6. root: refer to the audit log.

```
$ audit2why -a
```

```
type=AVC msg=audit(1455211133.736:523): avc:  denied  { transition } for  pid=4
255 comm="sshd" path="/bin/bash" dev="mmcblk0p3" ino=258580 scontext=system_u:s
ystem_r:sshd_t:s0-s0:c0.c1023 tcontext=sysadm_u:sysadm_r:sysadm_t:s0 tclass=pro
cess permissive=0
        Was caused by:
        The boolean ssh_sysadm_login was set incorrectly.
        Description:
        Allow ssh to sysadm login

        Allow access by executing:
        # setsebool -P ssh_sysadm_login 1
```

Figure 16. Audit log for sshd

```
$ audit2allow -a
```

Chapter 5
NETCONF/YANG

5.1 Overview

The NETCONF protocol defines a mechanism for device management and configuration retrieval and modification. It uses a remote procedure call (RPC) paradigm and a system of exposing device (server) capabilities, which enables a client to adjust to the specific features of any network equipment. NETCONF further distinguishes between state data (which is read-only) and configuration data (which can be modified). Any NETCONF communication happens on four layers as shown in the table below. XML is used as the encoding format.

Table 18. The NETCONF layers

Layer	Purpose	Example
1	Content	Configuration data, Notification data
2	Operations	<edit-config>
3	Messages	<rpc>, <rpc-reply>, <notification>
4	Secure	Transport SSH, TLS

YANG is a standards-based, extensible, hierarchical data modeling language that is used to model the configuration and state data used by NETCONF operations, remote procedure calls (RPCs), and server event notifications. The device configuration data are stored in the form of an XML document. The specific nodes in the document as well as the allowed values are defined by a model, which is usually in YANG format or possibly transformed into YIN format with XML-based syntax. There are many such models created directly by IETF to further support standardization and unification of the NETCONF interface of the common network devices. For example, the general system settings of a standard computer are described in the IETF-system model (rfc7317) or the configuration of its network interfaces defined by the IETF-interfaces model (rfc7223). However, it is common for every system to have some specific parts exclusive to it. In that case there are mechanisms defined to enable extensions while keeping the support for the standardized core. Also, as this whole mechanism is designed in a liberal fashion, the configuration does not have to concern strictly network. Even RPCs additional to those defined by NETCONF can be characterized, thus allowing the client to request an explicit action from the server.

A YANG module defines a data model through its data, and the hierarchical organization of and constraints on that data. A module can be a complete, standalone entity, or it can reference definitions in other modules and sub-modules as well as augment other data models with additional nodes. The module dictates how the data is represented in XML.

A YANG module defines not only the syntax but also the semantics of the data. It explicitly defines relationships between and constraints on the data. This enables you to create syntactically correct configuration data that meets constraint requirements and enables you to validate the data against the model before uploading it and committing it on a device.

For information about NETCONF, see RFC 6241, NETCONF Configuration Protocol.

For information about YANG, see RFC 6020, YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF), and related RFCs.

5.2 Netopeer

Netopeer is a set of NETCONF tools built on the libnetconf (https://github.com/CESNET/libnetconf) library. It allows developers to control their devices via NETCONF and operators to connect to their NETCONF-enabled devices.

5.2.1 libnetconf

libnetconf is a NETCONF library in C intended for building NETCONF clients and servers. It provides basic functions to connect NETCONF client and server to each other via SSH, to send and receive NETCONF messages and to store and work with the configuration data in a datastore. libnetconf implements the NETCONF protocol introduced by IETF. libnetconf is maintained and further developed by the Tools for Monitoring and Configuration department of CESNET.

For information about libnetconf see:

https://github.com/CESNET/libnetconf

https://rawgit.com/CESNET/libnetconf/master/doc/doxygen/html/index.html

5.2.2 Netopeer server

Netopeer software is a collection of utilities and tools to support the main application, Netopeer server, which is a NETCONF server implementation. It uses libnetconf for all NETCONF communication. Conforming to the relevant RFCs2 and still being part of the aforementioned library, it supports the mandatory SSH as the transport protocol but also TLS. Once a client successfully connects using either of these transport protocols and establishes a NETCONF session, it can send NETCONF RPCs and the Netopeer server responds with correct replies.

Most of the standard capabilities mentioned in the RFC 6241 are implemented including validating the new configuration before applying it or being able to return to a previous configuration if the new one failed to be applied. Except these, Netopeer server supports additional features such as these:

- Sending notifications on certain events to a client, provided that the client subscribes to them
- Access control, when every user has the available parts of the configuration for reading and for writing specified and cannot access any other.

The following set of tools are a part of the Netopeer server:

- Netopeer-server as the main service daemon integrating the SSH/TLS server.
- Netopeer-manager as a tool to manage the netopeer server modules.
- Netopeer-configurator as a tool for the server's first run configuration.

5.2.3 Netopeer client

Netopeer-cli is a CLI interface that allows you to connect to a NETCONF-enabled device and obtain and manipulate its configuration data.

This application is a part of the Netopeer software bundle, but compiled and installed separately. It is a NETCONF client with a command line interface developed and primarily used for Netopeer server testing, but allowing all the standards and even some optional features of a full-fledged NETCONF client.

Netopeer-cli serves as a generic NETCONF client providing a simple interactive command line interface. It allows you to establish a NETCONF session with a NETCONF-enabled device on the network and to obtain and manipulate its configuration data. Netopeer-cli is limited to a single NETCONF connection at a time via a forward or a reverse (Call Home) connecting method.

5.3 sja1105 YANG models

Following are the register tables in the sja1105 YANG model:

"schedule-table",

"schedule-entry-points-table",

"vl-lookup-table",

"vl-policing-table",

"vl-forwarding-table",

"l2-address-lookup-table",

"l2-policing-table",

"vlan-lookup-table",

"l2-forwarding-table",

"mac-configuration-table",

"schedule-parameters-table",

"schedule-entry-points-parameters-table",

"vl-forwarding-parameters-table",

"l2-address-lookup-parameters-table",

"l2-forwarding-parameters-table",

"clock-synchronization-parameters-table",

"avb-parameters-table",

"general-parameters-table",

"retagging-table",

"xmii-mode-parameters-table",

Each table owns its own registers list. The sja1105 YANG model tries to add all the table entry elements as 'leaves'.

Table 19. sja1105 YANG model tree

module: sja1105
Table continues on the next page...

Table 19. sja1105 YANG model tree (continued)

Configuration	+--rw sja1105 +--rw ptp l +--rw pin_duration? string l +--rw pin_start? string l +--rw schedule_time? string l +--rw schedule_correction_period? string l +--rw ts_based_on_ptpclk? string l +--rw schedule_autostart? string l +--rw pin_toggle_autostart? string +--rw static +--rw vl-lookup-table l +--ro name? string l +--rw entry* [index] l +--rw index uint32 l +--rw port? string l +--rw destports? string l +--rw iscritical? string l +--rw macaddr? string l +--rw vlanid? string l +--rw vlanprior? string l +--rw egrmirr? string l +--rw ingrmirr? string l +--rw vlid? string +--rw vl-policing-table l +--ro name? string l +--rw entry* [index] l +--rw index uint32 l +--rw type? string l +--rw maxlen? string l +--rw sharindx? string l +--rw bag? string l +--rw jitter? string +--rw vl-forwarding-table l +--ro name? string l +--rw entry* [index]

Table continues on the next page...

Table 19. sja1105 YANG model tree (continued)

```
| +--rw index uint32
| +--rw type? string
| +--rw priority? string
| +--rw partition? string
| +--rw destports? string
+--rw vl-forwarding-params-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw partspc? string
| +--rw debugen? string
+--rw avb-params-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw destmeta? string
| +--rw srcmeta? string
+--rw schedule-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw winstindex? string
| +--rw winend? string
| +--rw winst? string
| +--rw destports? string
| +--rw setvalid? string
| +--rw txen? string
| +--rw resmedia_en? string
| +--rw resmedia? string
| +--rw vlindex? string
| +--rw delta? string
+--rw schedule-entry-points-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
```

Table continues on the next page...

Table 19. sja1105 YANG model tree (continued)

| +--rw subschindx? string
| +--rw delta? string
| +--rw address? string
+--rw l2-address-lookup-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw vlanid? string
| +--rw macaddr? string
| +--rw destports? string
| +--rw enfport? string
+--rw l2-policing-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw sharindx? string
| +--rw smax? string
| +--rw rate? string
| +--rw maxlen? string
| +--rw partition? string
+--rw vlan-lookup-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw ving_mirr? string
| +--rw vegr_mirr? string
| +--rw vmemb_port? string
| +--rw vlan_bc? string
| +--rw tag_port? string
| +--rw vlanid? string
+--rw l2-forwarding-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw bc_domain? string

Table continues on the next page...

Table 19. sja1105 YANG model tree (continued)

```
| +--rw reach_port? string
| +--rw fl_domain? string
| +--rw vlan_pmap? string
+--rw mac-configuration-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw top? string
| +--rw base? string
| +--rw enabled? string
| +--rw ifg? string
| +--rw speed? string
| +--rw tp_delin? string
| +--rw tp_delout? string
| +--rw maxage? string
| +--rw vlanprio? string
| +--rw vlanid? string
| +--rw ing_mirr? string
| +--rw egr_mirr? string
| +--rw drpnona664? string
| +--rw drpdtag? string
| +--rw drpuntag? string
| +--rw retag? string
| +--rw dyn_learn? string
| +--rw egress? string
| +--rw ingress? string
+--rw schedule-parameters-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw subscheind? string
+--rw schedule-entry-points-parameters-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
```

Table continues on the next page...

Table 19. sja1105 YANG model tree (continued)

```
| +--rw clksrc? string
| +--rw actsubsch? string
+--rw l2-address-lookup-parameters-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw maxage? string
| +--rw dyn_tbsz? string
| +--rw poly? string
| +--rw shared_learn? string
| +--rw no_enf_hostprt? string
| +--rw no_mgmt_learn? string
+--rw l2-forwarding-parameters-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw max_dynp? string
| +--rw part_spc? string
+--rw general-parameters-table
| +--ro name? string
| +--rw entry* [index]
| +--rw index uint32
| +--rw vllupformat? string
| +--rw mirr_ptacu? string
| +--rw switchid? string
| +--rw hostprio? string
| +--rw mac_fltres1? string
| +--rw mac_fltres0? string
| +--rw mac_flt1? string
| +--rw mac_flt0? string
| +--rw incl_srcpt1? string
| +--rw incl_srcpt0? string
| +--rw send_meta1? string
| +--rw send_meta0? string
| +--rw casc_port? string
```

Table continues on the next page...

Table 19. sja1105 YANG model tree (continued)

	l +--rw host_port? string l +--rw mirr_port? string l +--rw vlmarker? string l +--rw vlmask? string l +--rw tpid? string l +--rw ignore2stf? string l +--rw tpid2? string +--rw xmii-mode-parameters-table l +--ro name? string l +--rw entry* [index] l +--rw index uint32 l +--rw phy_mac? string l +--rw xmii_mode? string
State Data	+--rw static +--rw version l +--ro deviceId? string l +--ro revision? string +--ro config-files l +--ro configFile* string +--ro ports +--ro port* string
RPC	rpcs: +---x save-local-config l +---w input l +---w configfile? string +---x load-local-config l +---w input l +---w configfile? string +---x load-default
Notification	-

5.4 Installing Netopeer-cli on Centos/Ubuntu

Use the following steps for installing Netopeer-cli on Centos or Ubuntu operating systems.

Steps for installation on Centos 7.2:

1. Install the following packages:

```
$ sudo yum install libxml2-devel libxslt-devel openssl-devel libgcrypt dbus-devel doxygen
libevent readline.x86_64 ncurses-libs.x86_64 ncurses-devel.x86_64 libssh.x86_64
libssh2-devel.x86_64 libssh2.x86_64 libssh2-devel.x86_64
```

2. Install pyang :

```
$ git clone https://github.com/mbj4668/pyang.git
$ cd pyang
$ sudo python setup.py install
```

3. Pull, configure, make, and install libnetconf:

```
$ git clone https://github.com/CESNET/libnetconf.git
$ cd libnetconf
$ ./configure
$ sudo make
$ sudo make install
```

4. Pull netopeer and configure, make, and install cli:

```
$ git clone https://github.com/CESNET/netopeer.git
$cd netopeer/cli
$ ./configure
$ make
$ make install
```

Steps for installation on Ubuntu 16.04

1. Install the following packages:

```
$ sudo apt-get install libxml2-dev libxslt-dev libssl-dev libssh2-1-dev xsltproc
        doxygen pkg-config libtool-bin cmake libssh-dev libevent-dev libreadline-dev
        libcurl4-openssl-dev python-libxml2 autoconf
```

2. Install pyang:

```
$ git clone https://github.com/mbj4668/pyang.git
$ cd pyang
$ sudo python setup.py install
```

NOTE

There is a version issue for libssh installation on Ubuntu below version 16.04. Apt-get install libssh may get version 0.6.4. But libnetconf needs a version of 0.7.3 or later. Remove the default one and reinstall by downloading the source code and installing it manually.

```
$ dpkg -r libssh-dev
$ wget https://red.libssh.org/attachments/download/195/libssh-0.7.3.tar.xz --no-check-
certificate
$ unxz libssh-0.7.3.tar.xz
$ tar -xvf libssh-0.7.3.tar
$ cd libssh-0.7.3/
$ mkdir build
$ cd build/
$ cmake ..
$ make
$ sudo make install
```

3. Pull, configure, make, and install libnetconf:

```
$ git clone https://github.com/CESNET/libnetconf.git
$ cd libnetconf
$ ./configure
$ sudo make
$ sudo make install
```

4. Pull netopeer and configure, make, and install cli:

```
$ git clone https://github.com/CESNET/netopeer.git
$ cd netopeer/cli
$ ./configure
$ make
$ sudo make install
```

5.5 Configuration

5.5.1 Netopeer-server

The netopeer-server is the NETCONF protocol server running as a system daemon. The netopeer-server is based on the libnetconf library. It provides an environment to run transAPI modules for configuring a specific device or application according to its data model.

- -d: Run in daemon mode.
- -h: Show help.
- -V: Show program version.
- -v: level Set the verbosity level. Possible values are from 0 (default) to 3. This overrides any NETOPEER_VERBOSE environment variable.

5.5.2 Netopeer-manager

The netopeer-manager provides access to the configuration of the netopeer-server modules. The netopeer-server modules extend its functionality to control another devices or applications via transAPI or just by storing configuration data. It can be configured using the options and commands described below.

OPTIONS

- **--help**: Prints the generic description and a list of commands. The detailed description and list of arguments for the specific command are displayed by using `--help` argument of the command.

COMMANDS

- **add**: Adds a new netopeer-server module. The added module is enabled by default and it is loaded by the netopeer-server during its next start. Use the following example:

```
add [--help] --name NAME (--model MODEL | --augment AUGMENT | --import IMPORT)
      [--transapi TRANSAPI] [--features FEATURE [FEATURE ...]] [--datastore DATASTORE]
```

  Where the arguments are as follows:

  - **--name** NAME Specifies the name of the netopeer-server module. The NAME is used as an identifier of the module in the netopeer-server configuration.

- **--model** MODEL Specifies path (absolute or relative) to the module's main data model in YIN format. In this option, the whole module configuration is created.
- **--augment** AUGMENT Specifies path (absolute or relative) to an augment model of the main data model in YIN format. This model is always appended at the end of the model list.
- **--import** IMPORT Specifies path (absolute or relative) to a model in YIN format that is imported by the main model. This model is always prepended at the beginning of the model list.
- **--transapi** TRANSAPI Optional parameter to specify path to the transAPI module related to the module's main data model. If the transAPI module is not specified, netopeer-server allows the connection and configuration data manipulation according to the data model, but the changes are not applied to any device. This part of the process is handled just by the trans API module.
- **--features** FEATURE [FEATURE ...] Data model can define various features that extend its basic functionality. By default, netopeer-server supposes all features to be disabled. This option explicitly specifies the list of features to enable. To enable all features, use the format `value *` .
- **--datastore** DATASTORE specifies the path to the file where the configuration data is stored. If not specified, the datastore is implemented as empty and it does not store any configuration data.

- **list**: Prints the list of all netopeer-server modules. The command format is as follows:

```
list [--help] [--name NAME]
```

  Where:

  - **--name**: specifies the name of the main netopeer-server module for which, the list of extending data models would be printed.

- **rm**: Removes the specified netopeer-server main module. The command format is as follows:

```
rm [--help] --name NAME [--model MODEL]
```

  Where the arguments are

  - --**name**: NAME Specifies the name of the main netopeer-server module to remove.
  - --**model**: If MODEL is specified, only this extending model is removed instead of the whole module.

5.5.3 netopeer-cli

The netopeer-cli is command line interface similar to the NETCONF client. It serves as a generic NETCONF client providing a simple interactive command line interface. It allows user to establish a NETCONF session with a NETCONF-enabled device on the network and to obtain and manipulate its configuration data. netopeer-cli is limited to a single NETCONF connection at a time via a forward or a reverse (Call Home) connecting method.

5.5.3.1 Netopeer CLI commands

Following are the Netopeer CLI commands:

1. **help**: Displays a list of commands. The `--help` option is also accepted by all commands to show detailed information about the command.
2. **Connect**: Connects to a NETCONF server.

```
connect [--help] [--login username] [--port num] host
```

   The **connect** command has the following arguments:

   - **--login** username: Specifies the user to log in as on the NETCONF server. If not specified, current local username is taken.

- **--tls**: Uses NETCONF over TLS transport instead of default SSH. Default client certificate and trusted CA directory are used for TLS handshake. This option is available only when the netopeer-cli is compiled with configure's `--enable-tls` option.
- **--cert** cert_path
  - Uses a specific certificate for TLS handshake. `cert_path` specifies path to the client certificate in CRT format. If --key option is not specified, `cert_path` is expected to contain also the private key for the client certificate, in PEM format.
  - This option is available only when the netopeer-cli is compiled with configure's `--enable-tls` option.
- **--key** key_path
  - Specifies path to the private key for the client certificate in KEY format. If not specified, cert_path is expected to contain also the private key for the client certificate, in PEM format.
  - This option is available only when the netopeer-cli is compiled with configure's `--enable-tls`option.
- **--trusted** trusted_CA_store
  - Specifies path to a trusted CA certificate bundle in PEM format to be used exclusively for server verification for this connection instead of the default CA directory.
  - This option is available only when the netopeer-cli is compiled with configure's `--enable-tls` option.
- **--port** num
  - Port to connect to on the NETCONF server. By default, port 830 for SSH or 6513 for TLS transport is used.
- **host**
  - Hostname of the target NETCONF server.

3. **disconnect**: disconnects from a NETCONF server.
4. **commit**
   - Performs the NETCONF `commit` operation. For details, see RFC 6241 section 8.3.4.1.
5. **copy-config**: Performs NETCONF `copy-config` operation. For details, see RFC 6241 section 7.3.

```
copy-config [--help] [--defaults mode] [--source datastore | --config file] target_datastore
```

   Where, the arguments are the following:

   - **--defaults** mode: Use: with the `-defaults` capability with specified retrieval mode. For details, refer to the RFC 6243 section 3 or WITH-DEFAULTS section of this manual.
   - **--source** datastore: Specifies the source datastore for the `copy-config` operation. For description of the datastore parameter, refer to Netopeer CLI datastore on page 69.

     NOTE

     This option is available only when the netopeer-cli is compiled with configure's `--enable-tls` option.

   - **--config** file
     - Specifies the path to a local file containing the complete configuration to copy. This option alternates the `--source` option.
   - **target_datastore**
     - Target datastore to be rewritten. For description of possible values, refer to Netopeer CLI datastore on page 69.

6. **delete-config** Performs NETCONF `delete-config` operation. For more details see RFC 6241 section 7.4.

```
delete-config [--help] target_datastore
```

Where

- **target_datastore** is the Target datastore to delete. For description of possible values, refer to Netopeer CLI datastore on page 69. Note that the running configuration datastore cannot be deleted.
- discard-changes: Performs NETCONF <discard-changes> operation. It reverts the candidate configuration to the current running configuration. For more details see RFC 6241 section 8.3.4.2.

7. **edit-config**

   Performs NETCONF `edit-config` operation. For details, see RFC 6241 section 7.2.

```
edit-config [--help] [--defop operation] [--error action] [--test
                        option] [--config file | --url URI] target_datastore
```

   Where

   - **--defop** operation
     - Specifies default operation for applying configuration data.
     - merge: Merges configuration data at the corresponding level. This is the default value.
     - replace: Edits configuration data completely replaces the configuration in the target datastore.
     - none: The target datastore is unaffected by the edit configuration data, unless and until the edit configuration data contains the operation attribute to request a different operation. For more info, see the EDIT-CONFIG section of this document.
   - **--error** action
     - Sets reaction to an error.
     - Stop: Aborts the operation on first error. This is the default value.
     - Continue: Continues to process configuration data on error. The error is recorded and negative response is returned.
     - Rollback: Stops the operation processing on error and restore the configuration to its complete state at the start of this operation. This action is available only if the server supports rollback-on-error capability (see RFC 6241 section 8.5).
   - **--test** option
     - Performs validation of the modified configuration data. This option is available only if the server supports `:validate:1.1` capability (see RFC 6241 section 8.6).
     - set: Does not perform validation test.
     - test-only: Does not apply the modified data, only perform the validation test.
     - test-then-set: Performs a validation test before attempting to apply modified configuration data. This is the default value.
   - **--config** file
     - Specify path to a file containing edit configuration data. The content of the file is placed into the `config` element of the edit-config operation. Therefore, it does not have to be a well-formed XML document with only a single root element. If neither --config nor --url is specified, user is prompted to write edit configuration data manually. For examples, see the EDIT-CONFIG section of this document.
   - **--url** URI
     - Specifies remote location of the file containing the configuration data hierarchy to be modified, encoded in XML under the element `config` in the `urn:ietf:params:xml:ns:netconf:base:1.0` namespace. Note, that this differs from file parameter, where the `config` element is not expected.
   - **target_datastore**

- Target datastore to modify. For description of possible values, refer to Netopeer CLI datastore on page 69. Note that the url configuration datastore cannot be modified.

8. **get**: Performs NETCONF `get` operation. Receives both the status as well as configuration data from the current running datastore. For more details see RFC 6241 section 7.7. The command format is as follows:

```
get [--help] [--defaults mode] [--filter [file]]
```

- **--defaults** mode
  - Use with the `-defaults` capability with specified retrieval mode. For more details see RFC 6243 section 3 or WITH-DEFAULTS section of this manual.
- **--filter** [file]
  - Specifies if the request will contain subtree filter (RFC 6241 section 6). The option is able to accept path to the file containing the filter specification. If the path is not specified, user is prompted to write the filter specification manually.
- **get-config** Performs NETCONF `get-config` operation. Retrieves only configuration data from the specified target_datastore. For details, refer to RFC 6241 section 7.1.

```
get-config [--help] [--defaults mode] [--filter [file]] target_datastore
```

- **--defaults** mode
  - Use: with the `-defaults` capability with specified retrieval mode. For more details see RFC 6243 section 3 or WITH-DEFAULTS section of this manual.
- **--filter** [file]
  - Specifies if the request will contain subtree filter (RFC 6241 section 6). The option is able to accept path to the file containing the filter specification. If the path is not specified, user is prompted to write the filter specification manually.
- **target_datastore**
  - Target datastore to retrieve. For description of possible values, refer to Netopeer CLI datastore on page 69. Note, that the url configuration datastore cannot be retrieved.

9. **get-schema**: Performs NETCONF `get-schema` operation that retrieves specified data model used by the server. This operation is available only if the server implements the YANG module for NETCONF monitoring. The list of available schemas can be retrieved from /netconf-state/schemas subtree via the get operation. For more details see RFC 6022 sections 3.1 and 4.

```
get-schema [--help] [--version version] [--format format] identifier
```

- **--version** version
  - Version of the requested schema.
- **--format** format
  - The data modeling language (format) of the requested schema. The default value is yang.
- **identifier**
  - Identifier for the schema list entry.

10. **lock**

Performs the NETCONF `lock` operation to lock the entire configuration datastore of a server. For details, see RFC 6241 section 7.5.

```
lock [--help] target_datastore
```

Where the

- **target_datastore**: specifies the target datastore to lock. For description of possible values, refer to Netopeer CLI datastore on page 69. Note, that the url configuration datastore cannot be locked.

11. **unlock**: Performs the NETCONF `unlock` operation to release a configuration lock, previously obtained with the `lock` operation. For more details see RFC 6241 section 7.6.

```
unlock [--help] target_datastore
```

where

- **target_datastore**: specifies the target datastore to unlock. For description of possible values, refer to Netopeer CLI datastore on page 69. Note, that the url configuration datastore cannot be unlocked.

12. **user-rpc**: Sends your own content in an RPC envelope. This can be used for RPC operations defined in data models not supported by the netopeer-cli.

```
user-rpc [--help] [--file file]
```

13. **file** file
    - Specifies a file containing NETCONF RPC operation in XML format. Only the NETCONF `rpc` envelope is added to the file content and then it is sent to a server. If the option is omitted, user is prompted to type the RPC content manually.
14. **verbose**
    - Enables/disables verbose messages.
15. **debug**
    - Enables/disables debug messages. Available only if the netopeer-cli is compiled with configure's `--enable-debug` option.
16. **quit**
    - Quits the program.

5.5.3.2 Netopeer CLI datastore

Following are the netopeer CLI datastores:

- **running**
  - This is the base NETCONF configuration datastore holding the complete configuration currently active on the device. This datastore always exists.
- **startup**
  - The configuration datastore holding the configuration loaded by the device when it boots. This datastore is available only on servers that implement the `:startup` capability.
- **candidate**
  - The configuration datastore that can be manipulated without impacting the device's current configuration and that can be committed to the running configuration datastore. This datastore is available only on servers that implement `:candidate` capability.
- **url**:URI
  - Refers to a remote configuration datastore located at URI. The file that the URI refers to contains the configuration data hierarchy to be modified, encoded in XML under the element `config` in the `urn:ietf:params:xml:ns:netconf:base:1.0` namespace. This datastore is available only on servers that implement the `:url` capability.

5.5.4 Operation examples

In the sja1105 YANG model, you may set any register of sj1105 or a whole register map by netopeer-cli.

Make sure the netconf server is running (check the netopeer-server is running at ls1021atsn).

If the netopeer-server is not running, input the below command to allow it to run (in verbose mode):

```
$ /usr/local/bin/netopeer-server -v 3
```

Test commands at netopeer-client (a CentOS/Ubuntu PC to run netopeer-cli as an example):

Start client software

```
$ netopeer-cli
```

Connect the netconf server ls1021atsn board (use the IP on ls1021atsn, here 10.193.20.53 is example):

```
$ Netconf> connect --port 830 --login root 10.193.20.53
```

Get staus datas of server

```
$ netconf> get
```

User can get daters what he wants by --filter option

```
$ netconf> get --filter
```

Then input the filtered node you want to see in the text editor:

```
<sja1105>
  <static>
    <ports/>
  </static>
</sja1105>
```

Get config daters what he wants in `running/candidate/startup` space:

```
$ netconf> get-config running
$ netconf> get-config candidate
$ netconf> get-config startup
```

You can also add --filter parameter to filter what you want to see.

You can use `discard-changes` command to copy data from `running` to `candidate` space.

```
$ netconf> discard-changes
```

NOTE

It is recommend to run `discard-changes` at the first instance when the board boots up. Refer the step No. 5 in Section Troubleshooting on page 72.

Use the `commit` command to copy data from ‘candidate’ to ‘running’ space.

```
$ netconf> commit
```

You can edit any elements in the tables or multiple tables elements for running/candidate/startup space.

```
$ netconf> edit-config candidate
```

NOTE

The ‘candidate’ modification only edits the candidate registers map. It does not actually modify this information in the sja1105 true registers.

```
$ netconf> edit-config running
```

The ‘running’ modification only edits the running registers map. It is really modifying the sja1105 true registers.

```
$ netconf> edit-config startup
```

The ‘startup’ modification only edits the startup registers map. It does not actually modify this information in the sja1105 true registers

Input below text into the editor as an example and wait for got 'OK' message:

```
sja1105 xmlns="http://nxp.com/ns/yang/tsn/sja1105"
```

```
<static>
<l2-policing-table>
<entry>
<index>1</index>
<sharindx>0x1</sharindx>
<smax>0x8000</smax>
<rate>0xFA00</rate>
<maxlen>0x5EE</maxlen>
<partition>0x0</partition>
</entry>
</l2-policing-table>
</static>
</sja1105>
```

The preceding example shows how to edit the 'index 1' entry in the l2-policing-table.

Refer to Operation examples on page 69 to check the YANG model of the sja1105 data design.

You can edit `config` a saved xml file with all of or part of the full yang model nodes list. You can refer the files under `board/nxp/ls1021atsn/rootfs_overlay/etc/sja1105/*.xml` for how to input edit-config content in the cross compiler buildroot folder.

```
$ ls board/nxp/ls1021atsn/rootfs_overlay/etc/sja1105/*.xml
policing.xml prioritizing.xml standard.xml
```

You can even check the server board to see the xml file that has been saved on the server:

```
$ netconf> get --filter
<sja1105><static><config-files/></static></sja1105>
```

So as an example of the edit-config command would be (suppose we are under buildroot source code root folder):

```
$ netconf> edit-config --config board/nxp/ls1021atsn/rootfs_overlay/etc/sja1105/policing.xml
candidate
```

- Use `running` option to directly edit in the physical registers of sja1105.
- Use `candidate` option to edit the register into candidate space.

You can copy data between the `running/candidate/startup` by using `copy-config` or source xml file in the client:

```
$ netconf> copy-config --source running startup
```

Here is an example of copying the running space date to startup space.

Three more RPC calls are provided in the sja1105 YANG model.

If user wants to save current registers mapping into a file (must extend with .xml. This 'save-local-config' command would save current 'running' space config registers value into the file.)

In the netopeer-cli shell, by `user-rpc` command:

```
$ netconf> user-rpc
```

Input below text into the editor (`nc_standard.xml` is an example file name):

```
<save-local-config xmlns="http://nxp.com/ns/yang/tsn/sja1105">
<configfile>
nc_standard.xml
</configfile>
</save-local-config>
```

Then, a file `nc_standard.xml` would save to the `/etc/sja1105/` directory in the server (board).

NOTE

You can get all saved xml files by 'get' command by 'config-files' filter.

If user wants to load a xml to set the sja1105 registers into `running` and `candidate` space, user can use rpc command 'load-local-config'. The xml file must list in the `/etc/sja1105/`. (Get file list by using the `get` command and with `config-files` filter.)

In the netopeer-cli shell, by 'user-rpc' command:

```
$ netconf> user-rpc
```

Use the commands below to achieve this (policing.xml is an example name):

```
<load-local-config xmlns="http://nxp.com/ns/yang/tsn/sja1105">
<configfile>
policing.xml
</configfile>
</load-local-config>
```

Users can load a default registers mapping mode (sja1105 as a normal switch mode).

In the netopeer-cli shell, you can do this by using the 'user-rpc' command:

```
$ netconf> user-rpc
```

Input the below text in the editor (policing.xml is an example name):

```
<load-default xmlns="http://nxp.com/ns/yang/tsn/sja1105" />
```

5.6 Troubleshooting

1. Connect fails at client side:

```
libnetconf ERROR: Remote host key changed, the connection will be terminated!
libnetconf ERROR: Checking the host key failed.
```

Fixing:

The reason is that the SSHD key changed at the server.

You need to input command ‘quit’ netopeer-cli first. Then remove ~/.ssh/ known_hosts and restart netopeer-cli.

2. Command error shows:

```
libnetconf ERROR: Input channel error (Socket error: Connection reset by peer)
user-rpc: receiving rpc-reply failed.

Closing the session.
```

Fixing:

Lost connection in few minutes later if not to communicate with server.

Reconnect server.

3. Run command at netopeer-server (board):

```
[$ /usr/local/bin/netopeer-server -v 3
```

Got error:

```
netopeer-server[216]: sock_listen: could not bind "::0" port 830 (Address already in use)
netopeer-server[216]: Server is not listening on any address!
```

Fixing:

There should another netopeer-server is running. Use command to check:

```
$ ps | grep netopeer-server
```

If there is a netopeer-server process:

```
$ /usr/local/bin/netopeer-server -d
```

4. Terminating netopeer-server.

The right way to terminate the server is by using Ctrl ^ C or the below command:

```
$ /etc/init.d/S90netconf stop
```

5. Get an empty candidate contend:

```
$ netconf> get-config candidate

Result:
```

Fixing:

It is recommend to input a ‘discard-changes’ command when the first time for netopeer-server boot up and connected by client.

```
$ netconf> discard-changes
```

6. When you operate command ‘edit-config’ ‘get-config’ got error:

```
NETCONF error: operation-failed (application) - There is no device/data that could be affected.
```

Fixing:

The reason is that the server was not properly terminated. If you killed the server several times then all this is stored in the shared memory and that many times on next server startup the candidate datastore is not replaced by the running datastore. Do not use kill -9 to terminate netopeer-server process. Use Ctrl ^ C in the terminal with netopeer-server (or you can send SIGINT instead SIGKILL). You should restart netopeer-server (Ctrl^ C or /etc/init.d/S90netconf restart)

```
$ /etc/init.d/S90netconf restart
```

Then, in the client netopeer-cli soon after connection:

```
$ netconf> discard-changes
```

7. The netopeer-server default to set nothing registers of sja1105 chip when server startup. If user wants to let netopeer-server auto set a series registers in sja1105 at netopeer-server boot up time, you need to set the registers value to the startup space. Below contend introduce how to set ‘startup’ space. Edit ‘startup’ space configuration.

   startup’ space configures are for setting sja1105 registers when netopeer-server start time. There are several ways to edit ‘startup’ space configures data. Here are the examples:

   From the ‘running’ space configure copy to ‘startup’ space:

   ```
   $ netconf> copy-config --source running startup
   ```

   Copy from client local file (policing.xml is example file name) to server ‘startup’ space:

   ```
   $ netconf> copy-config copy-config --config policing.xml startup
   ```

   Modify some sub-tree registers from client local file (policing.xml is example file name) to server ‘startup’ space:

   ```
   $ netconf> edit-config --config policing.xml startup
   ```

8. Validate command shows:

   ```
   NETCONF error: operation-not-supported (application) - Request could not be completed because the
   requested operation is not supported by this implementation.
   ```

   NOTE

   The ‘validate’ command is not supported in the current version of NETCONF.

9. Datastores dead lock: When the server crashes or is terminated with SIGKILL, it may happen that the internal datastore locks stay locked. In such a case, the next time the netopeer-server (or any other libnetconf based application) tries to access the configuration datastores, it freezes. To solve this problem, release the locks manually removing the /dev/shm/sem.NCDS_FLOCK_* files.

   Refer to the point No. 4 above, for how to terminate the netopeer-server properly.

10. Modifying /etc/network/interfaces might cause the netopeer-server segment fault when netopeer-server starts if ietf-interfaces is selected in the buildroot. Use the below command to remove the ietf-interfaces model.

    ```
    $ /usr/local/bin/netopeer-manager rm --name ietf-interfaces
    $ /usr/local/bin/netopeer-manager list
    ```

Chapter 6
OPC UA

OPC (originally known as “OLE for Process Control”, now “Open Platform Communications”) is a collection of multiple specifications, most common of which is OPC Data Access (OPC DA).

OPC Unified Architecture (OPC UA) was released in 2010 by the OPC Foundation as a backward incompatible standard to OPC Classic, under the name of IEC 62541.

OPC UA has turned away from the COM/DCOM (Microsoft proprietary technologies) communication model of OPC Classic, and switched to a TCP/IP based communication stack (asynchronous request/response), layered into the following:

- Raw connections
- Secure channels
- Sessions

6.1 OPC introduction

OPC UA defines:

- The transport protocol for communication (that can take place over HTTP, SOAP/XML or directly over TCP).
- A set of 37 'services' that run on the OPC server, and which clients call into, via an asynchronous request/response RPC mechanism.
- A basis for creating information models of data using object-oriented concepts and complex relationships.

The primary goal of OPC is to extract data from devices in the easiest way possible.

The *Information Model* provides a way for servers to not only provide data, but to do so in the most self-explanatory and intuitive way possible.

NOTE

Further references to 'OPC' in this document will imply OPC UA. OPC Classic is not discussed in this document.

Following are the typical scenarios for embedding an OPC-enabled device into a project:

- Manually investigate (“browse”) the server’s Address Space looking for the data you need using a generic, GUI client (such as UaExpert from Unified Automation, or the FreeOpcUa covered in this chapter).
- Using References and Attributes, understand the format it is in, and the steps that may be needed to convert the data.
- Have a custom OPC client (integrated into the application) subscribe directly to data changes of the node that contains the desired data.

In a typical use case:

- The OPC server runs near the source of information (in industrial contexts, this means near the physical process – for example, on a PLC on a plant floor).
- Clients consume the data at run time (for example, logging into a database, or feeding it into another industrial process).

OPC-enabled applications can be composed: an industrial device may run an OPC client and feed the collected data into another physical process, while also exposing the latter by running an OPC server.

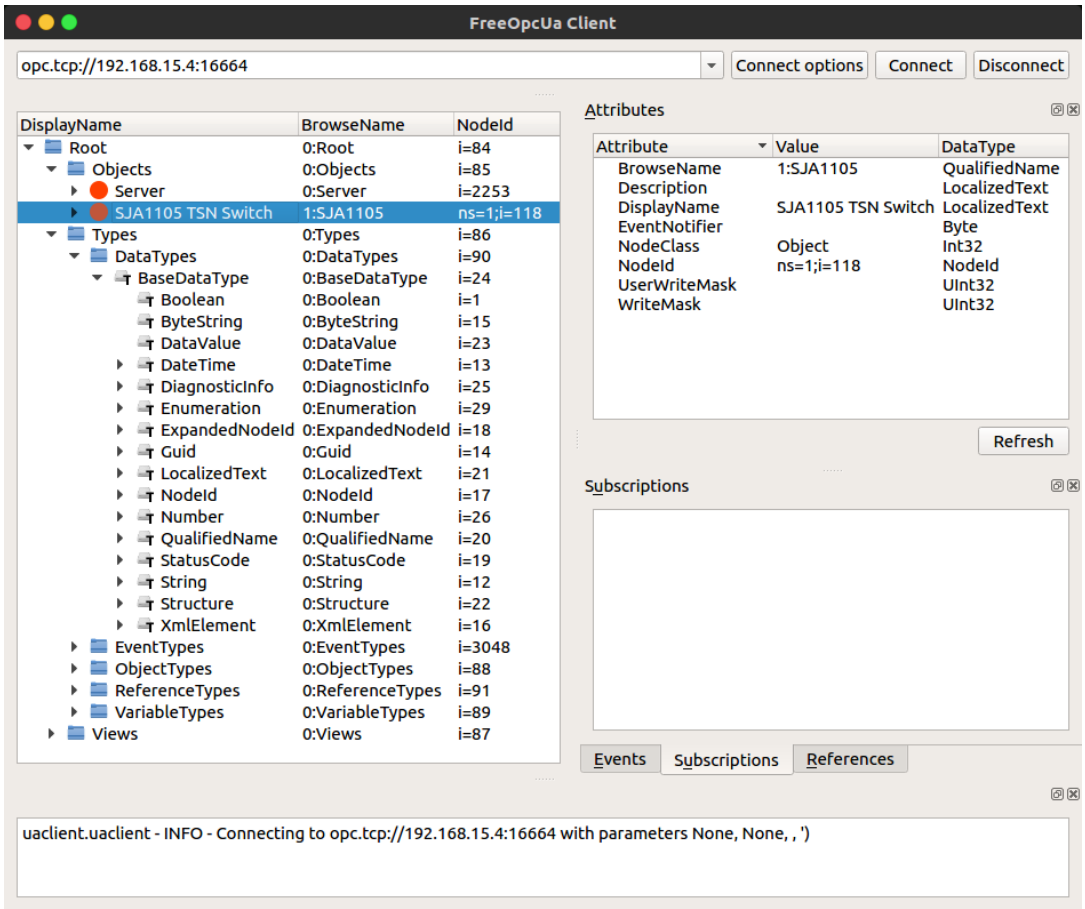
6.2 The node model

Data in an OPC server is structured in *Nodes*. The collection of all nodes that an OPC server exposes to its clients is known as an *Address Space*. Some nodes have a predefined meaning, while others have meaning that is unique to the *Information Model* of that specific OPC server.

Every Node has the following ***Attributes***:

- an ***ID*** (unique)
- a ***Class*** (what type of node it is)
- a ***BrowseName*** (a string for machine use)
- a ***DisplayName*** (a string for human use)

Figure 17. OPC UA address space



Shown on the left-hand side of the figure is the *Address Space* (collection of information that the server makes available to clients) of the OPC server found at `opc.tcp://192.168.15.4:16664`.

Selected is a node with NodeID `ns=1;i=118`, BrowseName=`1:SJA1105` and of NodeClass `Object`.

The full path of the selected node is `0:Root,0:Objects,1:SJA1105`.

6.3 Node Namespaces

Namespaces are the means for separating multiple Information Models present in the same Address Space of a server.

- Nodes that do not have the `ns=` prefix as part of the NodeID have an implicit `ns=0;` prefix (are part of the namespace `zero`).
- Nodes in *namespace * 0* have NodeID's pre-defined by the OPC UA standard. For example, the `0:Server` object, which holds self-describing information (capabilities, diagnostics, and vendor information), has a predefined NodeID of `ns=0;i=2253;`.

It is considered a good practice to not alter any of the nodes exposed in the *namespace * 0*.

6.4 Node classes

OPC nodes have an inheritance model, based on their *NodeClass*.

There are eight base node classes defined by the standard:

- Object
- Variable
- Method
- View
- ObjectType
- VariableType
- ReferenceType
- DataType

All nodes have the same base Attributes (inherited from the Node object), plus additional ones depending on their *NodeClass*.

6.5 Node graph and references

It may appear that nodes are only chained hierarchically, in a simple parent-child relationship. However, in reality nodes are chained in a complex directed graph, through *References* to other nodes.

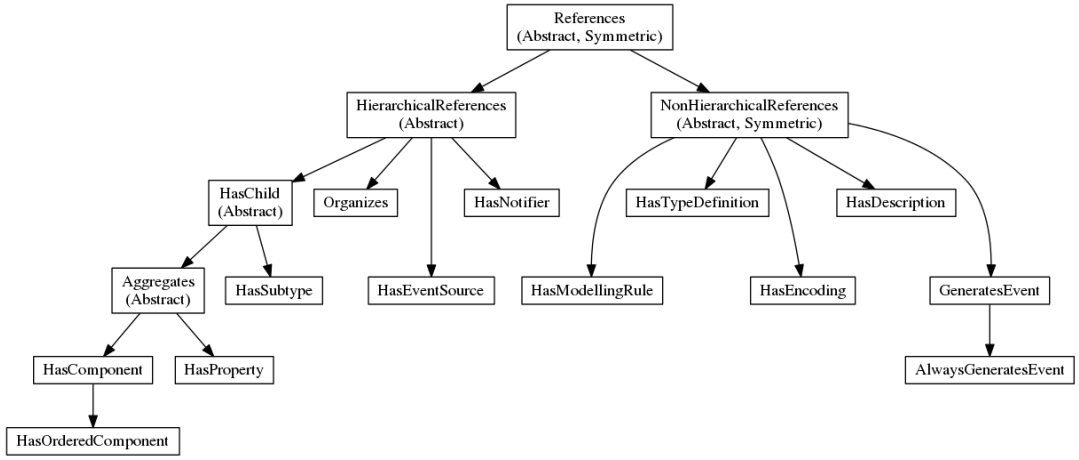


Figure 18. Hierarchy of the standard ReferenceTypes, defined in Part 3 of the OPC UA specification (Image taken from www.open62541.org)

In OPC, even ReferenceTypes are Nodes, and as such are structured hierarchically, as can be seen in the figure above.

The definitions of all OPC ReferenceTypes can be found under the `0:Root,0:Types,0:ReferenceTypes` path.

The semantics of OPC references can be enriched by creating custom ReferenceType nodes.

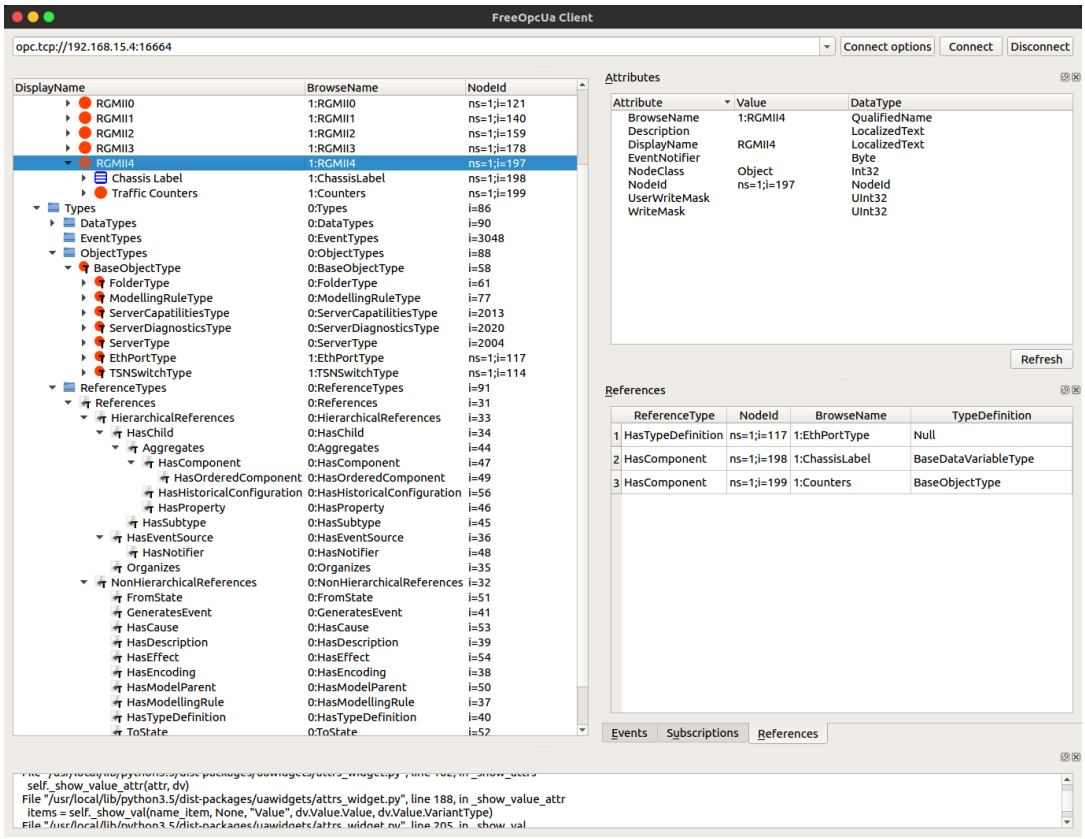


Figure 19. The 'Attributes' and 'References' views of the FreeOpcUa Client populated with details of the RGMII4 node

Selected in the Address Space is node `ns=1;i=197`. Conceptually, this represents one of the five Ethernet ports of the SJA1105 TSN switch.

Its NodeClass is Object, but it has a reference of type `HasTypeDefinition` to NodeID `ns=1;i=117` which is `1:EthPortType`. For this reason, the `1:RGMII4` node is of the custom ObjectType `EthPortType`.

6.6 Open62541

OpenIL integrates the Open62541 software stack (https://open62541.org/). This supports both server-side and client-side API for OPC UA applications. Only server-side capabilities of open62541 are being shown here.

Open62541 is distributed as a C-based dynamic library (libopen62541.so). The services run on pthreads, and the application code runs inside an event loop.

When building with the BR2_PACKAGE_OPEN62541_EXAMPLES flag, the following Open62541 example applications are included in the OpenIL target image:

- open62541_client
- open62541_server_instantiation
- open62541_tutorial_client_firststeps
- open62541_tutorial_server_firststeps
- open62541_tutorial_server_variable
- open62541_server
- open62541_server_mainloop
- open62541_tutorial_datatypes
- open62541_tutorial_server_method
- open62541_tutorial_server_variabletype
- open62541_server_inheritance
- open62541_server_repeated_job
- open62541_tutorial_server_datasource
- open62541_tutorial_server_object

6.7 Example of a server application: OPC SJA1105

In addition to the default Open62541 examples, OpenIL includes an application for monitoring the SJA1105 traffic counters on the LS1021A-TSN board. It can be started by running:

```
[root@openil] $ /usr/bin/opc-sja1105
```

The application's information model hierarchically describes the per-port traffic counters of the L2 switch under the `1:SJA1105` node.

On the server, a repeated job runs once per second, reads the port counters over SPI, and manually updates the port counter nodes.

6.8 FreeOpcUa Client GUI

FreeOpcUa (http://freeopcua.github.io/) is another open source framework for OPC UA communication (both server- and client-side). For this example, the client GUI available at https://github.com/FreeOpcUa/opcua-client-gui can be used to interact with the `opc-sja1105` server application from OpenIL.

1. Follow the instructions from the `opcua-client-gui README.md` to install it on a host PC (either Windows or GNU/Linux). As noted, a Python runtime with Qt5 support is required.
2. In Windows, navigate to the location of your WinPython installation, and open `WinPython Command Prompt.exe`.
3. Execute the following command:

   ```
   opcua-client
   ```

   The FreeOpcUa client GUI window pops up.
4. In the address drop-down input field, insert the following text:

   ```
   opc.tcp://192.168.15.2:16664
   ```

   After selecting **Connect**, a connection to the OPC UA server running on Board 2 is established.
5. In the OPC UA client, navigate to the node **Root -> Objects -> SJA1105 TSN Switch -> RGMII2 -> Traffic Counters -> ETH3 ::: N_TXBYTE**. This should correspond to the Node ID `ns=1;i=173`. Right click on this node, and select **Subscribe to data change**.
6. After this step, the OPC UA client should look like this:

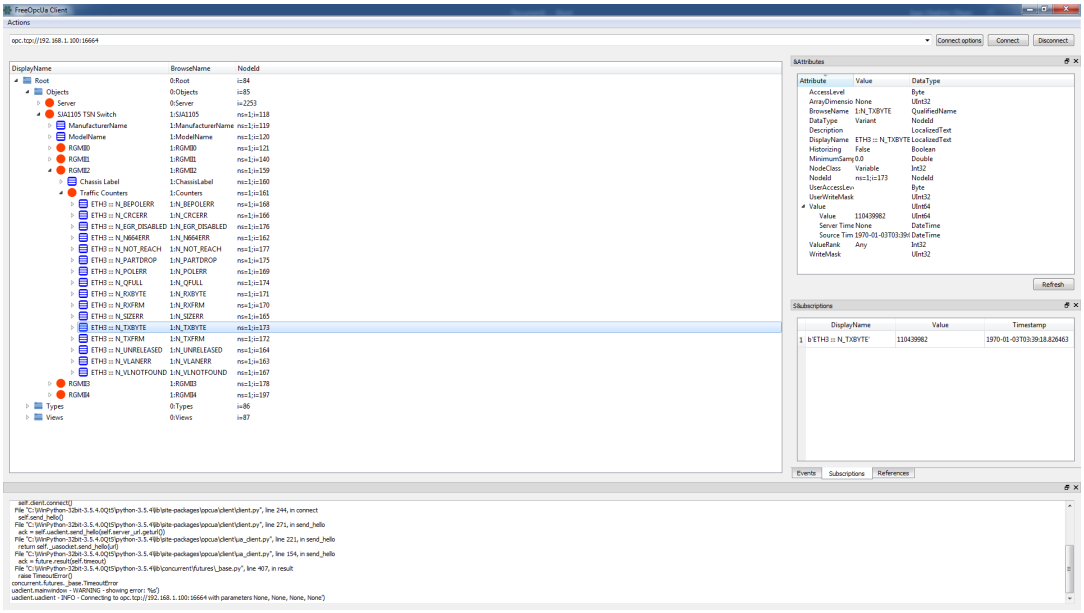


Figure 20. Subscribing to data changes of the ETH3 ::: N_TXBYTE node of the OPC-SJA1105 server

In the FreeOpcUa GUI, it is possible to create subscriptions to Data Changes on port counters of interest (by right-clicking on the individual nodes in the Address Space).

A dedicated OPC client might run custom code upon receiving Data Change notifications from the server, whereas the FreeOpcUa GUI only displays the updated values.

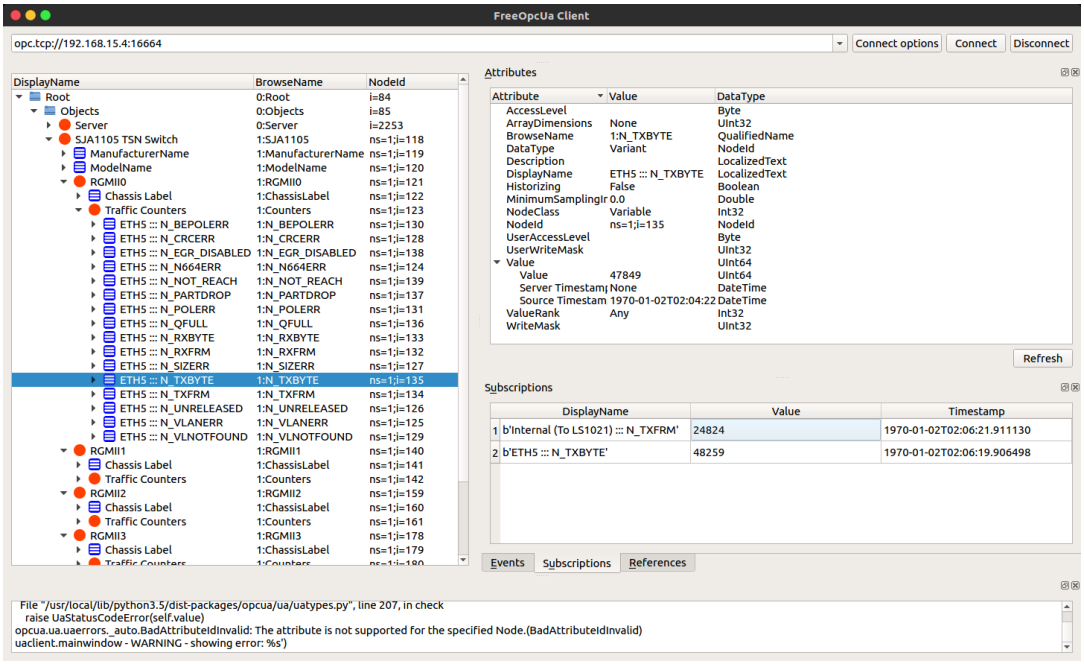


Figure 21. Data change notification

The preceding figure shows the Data Change Subscriptions to two counters: the Tx Frames of the L2 switch towards the LS1021, and the Tx Bytes towards chassis port ETH5.

Note that the subscribed value of `ETH5 ::: N_TXBYTE` (48259) is higher than the Value of its Attribute (47849). This is because the Subscriptions view updates automatically, while the Attributes do not.

Chapter 7
TSN Demo

7.1 Introduction

Time Sensitive Networking (TSN) is an extension to traditional Ethernet networks, providing a set of standards compatible with IEEE 802.1 and 802.3. These extensions are intended to address the limitations of standard Ethernet in sectors ranging from industrial and automotive applications to live audio and video systems.

Applications running over traditional Ethernet must be designed very robust in order to withstand corner cases such as packet loss, delay or even reordering. TSN aims to provide guarantees for deterministic latency and packet loss under congestion, allowing critical and non-critical traffic to be converged in the same network.

On the OpenIL platforms, TSN features are provided by the SJA1105TEL Automotive Ethernet switch present on the LS1021ATSN board. These hardware features can be used to implement the following IEEE standards:

- 802.1Qbv - Time Aware Shaping
- 802.1Qci - Per-Stream Filtering and Policing
- 1588v2 - Precision Time Protocol

There are two separate use cases being shown as part of this TSN demonstration:

- Rate limiting
- Synchronized Qbv

Both these use cases require a common topology comprising three LS1021ATSN boards and a host PC.

7.2 Bill of Materials

For the TSN demo, the Bill of Materials contains:

- 3 LS1021ATSN boards
- 1 host PC or laptop running Windows or a GNU/Linux distribution
- 1 regular L2 switch with 4 Ethernet ports
- Cabling, power adapters, microSD cards

7.3 Topology

The TSN demo topology consists of three IP networks :

- The **management network** (192.168.15.0/24) contains the eth0 interfaces of the three LS1021ATSN boards, connected together through the regular L2 switch, and to the host PC which has a static IP address of 192.168.15.100.
- The **untagged TSN network** (172.15.0.0/24), where the three participating network interfaces are the eth2 ports of each board (the Ethernet interface connected internally to the SJA1105 switch).
- The **tagged TSN network** (172.15.100.0/24) physically overlaps with the untagged network, but packets are transmitted with a VLAN ID of 100.

This topology is depicted in the following figure.

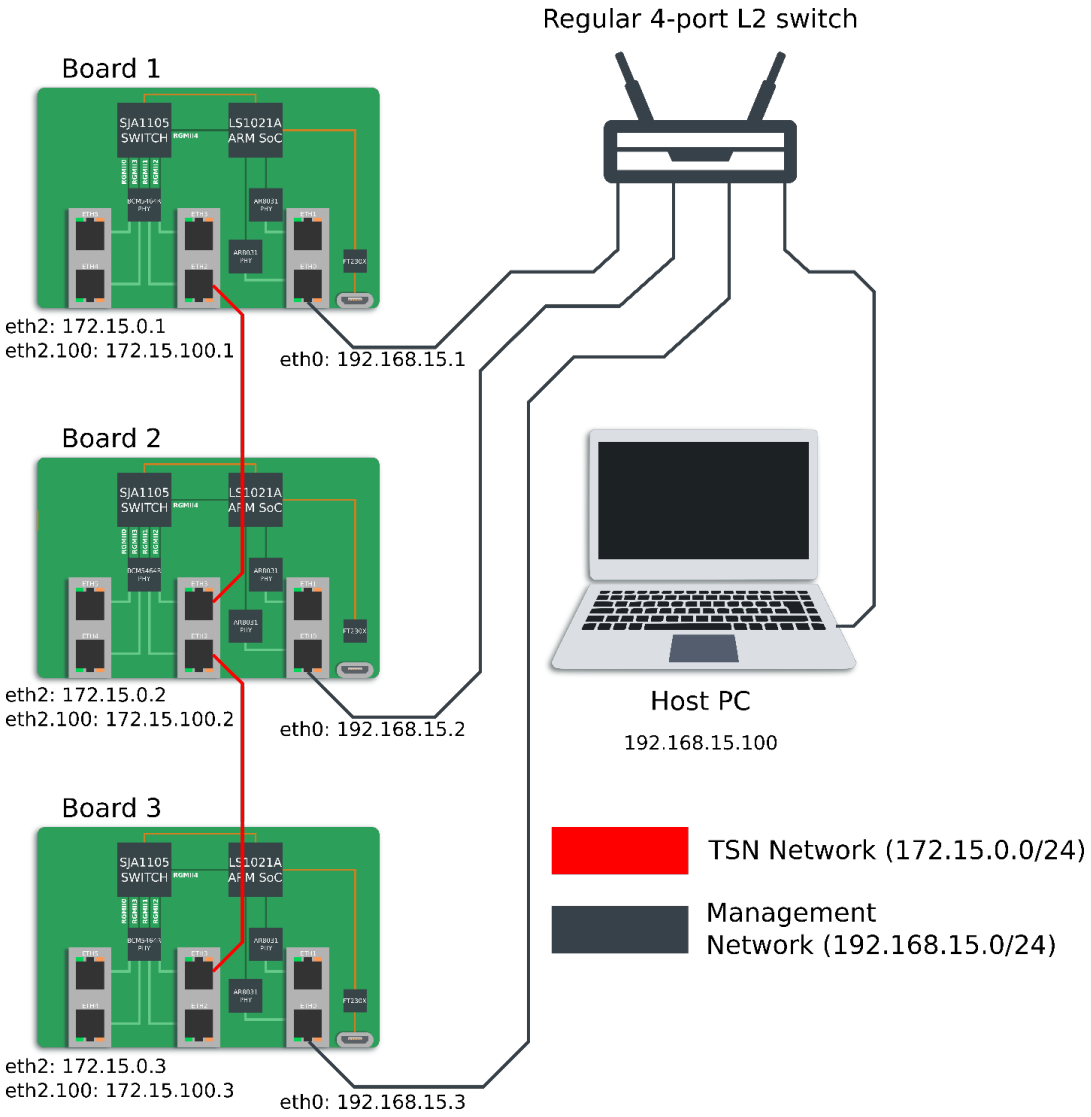


Figure 22. Topology of the demo network

7.4 Running the demo with a single LS1021ATSN board

Out of the two upcoming use cases of the TSN demo, only the Synchronized Qbv strictly requires the use of three LS1021ATSN boards. For the rate-limiting use case, a single LS1021ATSN board is sufficient (Board 2).

Boards 1 and 3 can be replaced with other hosts, which satisfy the following criteria:

- Have a 1 Gbps Ethernet port (that will replace the eth2 port of the LS1021 board)
- The Ethernet port is configured for a static IP in the 172.15.0.0/24 network (untagged TSN)
- Are able to run a GNU/Linux environment
- Have the following packages installed:
  - openssh-server
  - openssh-client
  - iperf3

- tcpdump
- prl (Pipe Rate-Limiter: https://github.com/openil/openil/tree/master/package/prl)

- Board 2 is able to log into these hosts via SSH without asking for password.
  - For key-based authentication, the public RSA key of Board 2 (`/etc/ssh/ssh_host_rsa_key.pub` in OpenIL) must be copied into the `~/.ssh/authorized_hosts configuration` file of the GNU/Linux user that LBT would attempt to log in.

NOTE

- The VLAN-tagged TSN network is not needed for the rate-limiting use case, so do not configure it.
- If Board 1 and Board 3 are replaced with PCs or laptops, there is no longer a need for having the L2 switch and the `192.168.15.0/24` management network.
- The LBT web application must be run by connecting to http://172.15.0.2:8000
- In OpenIL (Board 2), edit the `/usr/lib/node_modules/lbt/config.json` file and replace this line:

```
"measurementInterface": "eth2"
```

With the actual interface name of the Ethernet port of the TSN board replacement.

- Porting the `prl` program to the replacement hosts most likely involves compiling it from source and adding it to /usr/bin by running `make install`. If this is not desirable, view the `/usr/lib/node_modules/lbt/README.md` for instructions on how to patch the LBT application as to not require the prl program as dependency, and the drawbacks of doing so.

NOTE

NXP provides a separate document that describes how to run the rate-limiting TSN demo using one LS1021A-TSN board, a bootable live Ubuntu USB image, and a FRDM-LS1012A board. For more details about this setup, refer to *LS1021A TSN 1Board Demo Quick Start Guide*.

7.5 Host PC configuration

Required software:

- An SSH client (openssh-client for GNU/Linux; PuTTY, MobaXTerm, TeraTerm etc. for Windows)
- An application for serial communication (screen, minicom, etc. for GNU/Linux; PuTTY, MobaXTerm, TeraTerm etc. for Windows)

Follow these steps if you have a **Windows PC**:

1. On the PC, assign the static IP address of 192.168.15.100 to the Ethernet interface that is connected directly to the L2 switch, and indirectly to the 3 boards. In Windows, this is done by editing the network settings from the Control Panel as in the figure below:

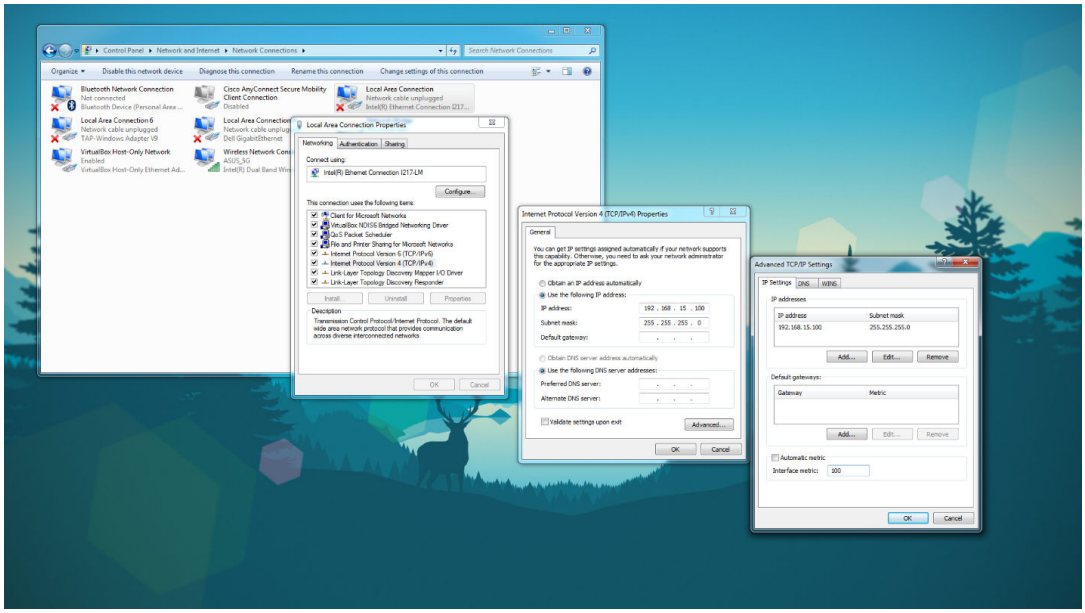


Figure 23. Windows network settings

2. Go to **Advanced settings** (in the same window) and configure a manual metric of 100 for this interface. This way, Internet traffic should not be routed through the Ethernet connection if the laptop is also connected to Wi-Fi.

Follow the below steps if you have a **GNU/Linux PC**:

1. Open GNOME NetworkManager by clicking on its system tray icon and then choosing "**Edit Connections…**" from the drop-down menu.
2. Choose your Ethernet interface from the dialog box, click "Edit" and go to the IPv4 Settings tab.
3. Change the Method to Manual and Add an IP address of 192.168.15.100.
4. Press the "Routes…" button and tick the "Use this connection only for resources on its network" checkbox.
5. Close all NetworkManager windows and click again on the system tray icon, this time selecting the newly configured Ethernet interface, in order to reload its configuration.

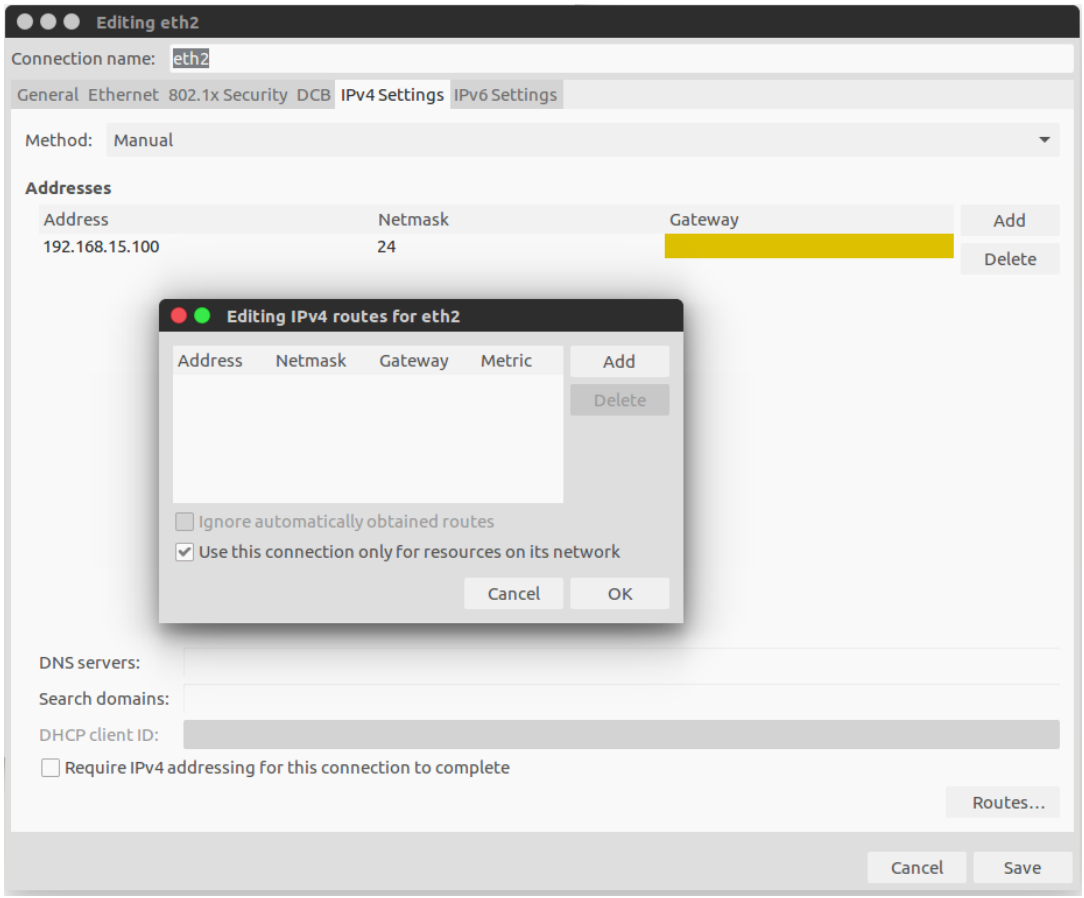


Figure 24. Ubuntu network settings (GNOME NetworkManager)

Connection to the three boards over SSH can be verified using MobaXTerm for Windows PC only:

1. Open MobaXTerm, click **Session -> SSH** and type in the IP address of Board 1: “192.168.15.1” with username “root”.

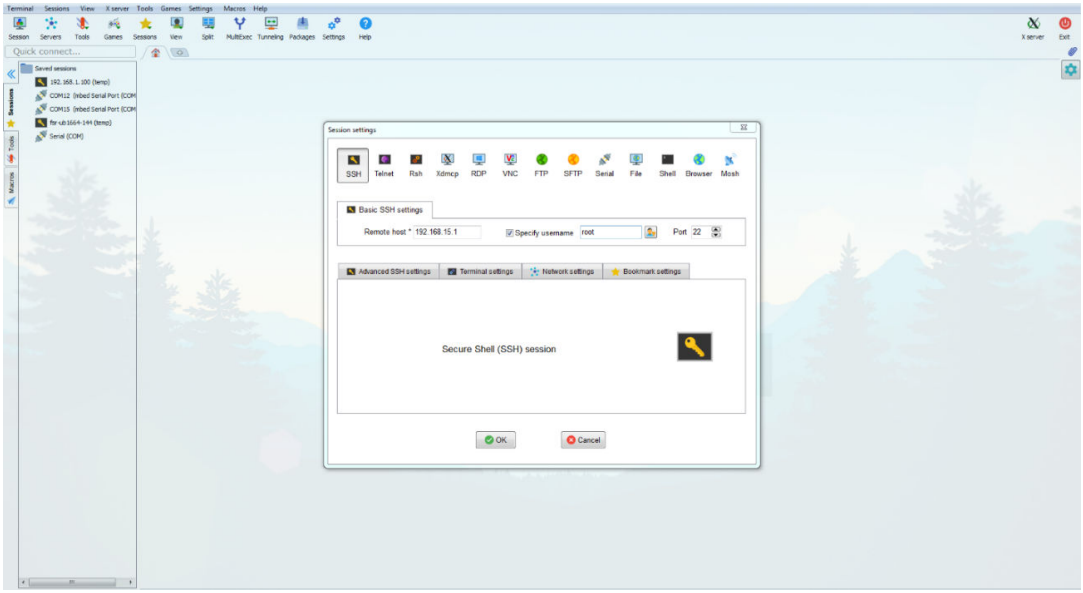


Figure 25. MobaXTerm session settings

2. Connect to Board 1 via MobaXTerm. You should be able to view this interface:

Figure 26. Login shell to OpenIL on Board 1 in MobaXTerm

3. Repeat the previous steps for Board 2 and Board 3.

7.6 Hardware Setup

Perform the following steps for each board individually (without having the whole topology assembled). You will need a serial connection.

1. Boot the boards to U-Boot. Then, change the MAC addresses of the three boards by executing these U-boot commands:

   Board 1:

   ```
   => setenv ethaddr  00:04:9f:ef:00:00
   => setenv eth1addr 00:04:9f:ef:01:01
   => setenv eth2addr 00:04:9f:ef:02:02
   ```

   Board 2:

   ```
   => setenv ethaddr  00:04:9f:ef:03:03
   => setenv eth1addr 00:04:9f:ef:04:04
   => setenv eth2addr 00:04:9f:ef:05:05
   ```

   Board 3:

   ```
   => setenv ethaddr  00:04:9f:ef:06:06
   => setenv eth1addr 00:04:9f:ef:07:07
   => setenv eth2addr 00:04:9f:ef:08:08
   ```

2. Resume the boot process for each board:

   ```
   => saveenv
   => boot
   ```

3. In Linux, modify the following files, by adapting the address to the board number (172.15.0.{1,2,3}, 172.15.100.{1,2,3}, 192.168.15.{1,2,3}):

/etc/network/interfaces (customize for boards 1, 2, 3):

```
# /etc/network/interfaces – configuration file for ifup(8), ifdown(8)
# The loopback interface
auto lo
iface lo inet loopback
auto eth2
iface eth2 inet static
address 172.15.0.1
netmask 255.255.255.0
auto eth1
iface eth1 inet dhcp
auto eth0
iface eth0 inet static
address 192.168.15.1
netmask 255.255.255.0
```

/etc/init.d/S45vlan (customize for boards 1, 2, 3):

```
IPADDR=”172.15.100.1/24”
```

/etc/hosts (copy as-is for boards 1, 2, 3):

```
127.0.0.1       localhost
127.0.1.1       OpenIL
172.15.0.1      board1
172.15.0.2      board2
172.15.0.3      board3
172.15.100.1    board1-vlan
172.15.100.2    board2-vlan
172.15.100.3    board3-vlan
192.168.15.1    board1-mgmt
192.168.15.2    board2-mgmt
192.168.15.3    board3-mgmt
192.168.15.100  host-pc
```

4. Now, disconnect the USB-serial cables and assemble the 3 boards, PC, and L2 switch in their final position:
5. Place the 3 LS1021ATSN boards in a stack, with Board 1 on top and Board 3 at the bottom.
6. Make the following connections between the 3 boards:
   a. Board 1 ETH2 to Board 2 ETH3
   b. Board 2 ETH2 to Board 3 ETH3
   c. Board 1 ETH0 to Regular L2 Switch
   d. Board 2 ETH0 to Regular L2 Switch
   e. Board 3 ETH0 to Regular L2 Switch
   f. Laptop to Regular L2 Switch

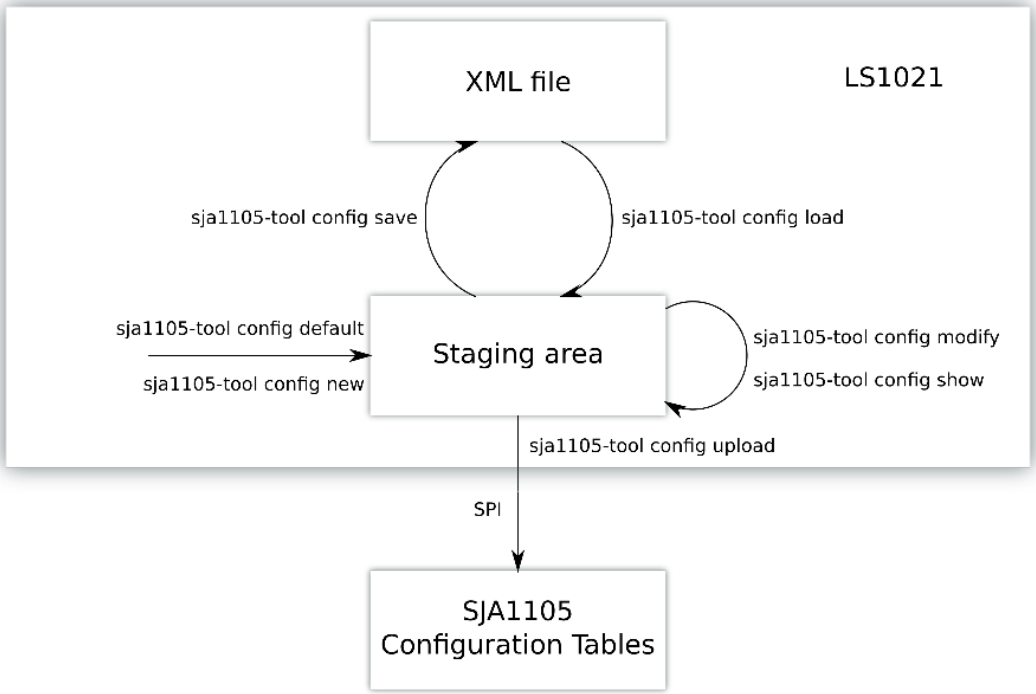
7.7 Managing configurations with the sja1105-tool

The `sja1105-tool` is a Linux user space application for configuring the SJA1105 TSN switch. The tool supports:

- Importing a configuration for the SJA1105 switch from an XML file

- Exporting the current SJA1105 configuration as an XML file
- Uploading the current SJA1105 configuration to the switch through its SPI interface
- Inspecting the current SJA1105 configuration
- On-the-fly modification of the current SJA1105 configuration through command line or scripting interface

Figure 27. State machine view of the sja1105-tool commands and configuration formats



The physical SPI registers of the SJA1105 switch are write-only. Therefore, a copy of these registers is kept in the staging area, which is effectively a file in the LS1021A OpenIL filesystem. The staging area keeps a binary image of the configuration to be uploaded over SPI using `sja1105-tool config upload`, and can also be read back by the user with `sja1105-tool config show`.

More documentation on the sja1105-tool is distributed as man pages along with the source code:

```
[ubuntu:~] $ git clone https://github.com/openil/sja1105-tool.git
[ubuntu:~] $ cd sja1105-tool
[ubuntu:sja1105-tool] $ cd docs/man
[ubuntu:man] $ man -l ./sja1105-tool.1
[ubuntu:man] $ man -l ./sja1105-tool-config.1
[ubuntu:man] $ man -l ./sja1105-tool-status.1
[ubuntu:man] $ man -l ./sja1105-tool-reset.1
[ubuntu:man] $ man -l ./sja1105-conf.5
[ubuntu:man] $ man -l ./sja1105-tool-config-format.5
```

7.7.1 SJA1105-tool helper scripts

In order to create other customized configurations, the sja1105-tool may be used as a host tool (run in a GNU/Linux userspace environment on a PC) and may generate XML files.

Install dependencies for sja1105-tool and its helper scripts (shown here for Ubuntu 16.04):

```
sudo apt-get install build-essential jq libxml2-dev
```

Set up sja1105-tool for host usage:

```
[ubuntu:~] $ git clone git@github.com:openil/sja1105-tool.git
 [ubuntu:~] $ cd sja1105-tool/src/helpers
 [ubuntu:~] $ source envsetup
```

Inside the sja1105-tool source files, there are two helper scripts in the src/helpers/bin/ folder:

- scheduler-create
- policer-limit

The two helper scripts above constitute the recommended high-level way of interacting with the SJA1105 ingress policer and Qbv engine. The input to the **policer-limit** script is provided through command-line arguments, while the **scheduler-create** script expects to read a JSON description of Qbv cycles and timeslots.

Actual examples of using the helper scripts are the files in the **src/helpers/configs/rate-limiting/** folder. These represent the four configurations presented in this use case and are named:

- standard.sh
- prioritizing.sh
- policing.sh
- scheduling.sh

Running each of these scripts produces an XML configuration of the same name that can be uploaded to Board 2 and loaded into the sja1105-tool.

7.8 Latency and bandwidth tester

Latency and bandwidth tester (LBT) is a web application written in Node JS that is distributed with the OpenIL image for LS1021ATSN and as such, runs on each of the three boards.

It serves as a web interface on the HTTP port 8000, through which users may configure the parameters of a traffic test. The two types of network tests it can perform are iPerf3 (for bandwidth measurements) and ping (for latency measurements). For each of these two types of traffic, users can define an arbitrary amount of flows (their source, destination, and flow-specific parameters).

After the configuration phase is finished and the traffic is started, the web server automatically connects over SSH to the source and destination host of each flow (ping and iPerf). The web server collects network traffic information in real-time from the source and destination hosts, and plots it inside the browser window.

For the following two use cases, we use the LBT web application to generate traffic and characterize the behavior of the SJA1105 TSN switch under different loads and configurations.

1. From a shell connected to any of the three LS1021ATSN boards, the LBT server can be started by running the following command:

   ```
   [root@openil] $ /usr/lib/node_modules/lbt/server.js
   ```

2. A successful invocation of the server displays this as the final line:

   ```
   Server listening for http requests on port 8000
   ```

3. If the server displays the following on the final line of its output, it means that the LBT server was already started and is running:

   ```
   Error: listen EADDRINUSE :::8000
   ```

4. In order to kill current nodes and start it again, run the command:

```
killall node
```

5. In a browser window, navigate to the management IP (in the 192.168.15.0/24 network) of the board, on port 8000, in order to view the application.

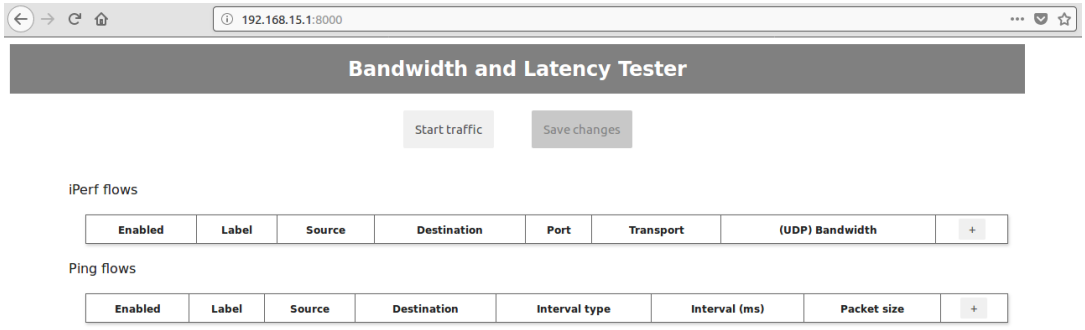


Figure 28. Latency and bandwidth tester default interface

As shown in the above figure, the traffic is stopped, and the two tables with iPerf and ping flows are empty. New flows can be added to the tables by pressing the “+” button.

NOTE

Traffic generation using LBT is absolutely equivalent, from an expected performance perspective, to running 'iperf' and 'ping' commands manually between boards from the command line.

The LBT web application has a configuration file under the following path:

`/usr/lib/node_modules/lbt/config.json.`

7.9 Rate limiting demo

7.9.1 Demo overview

The rate-limiting demo focuses on configuring the QoS features of a single SJA1105 switch, as to handle the congestion created by two competing traffic flows.

The use case conceptually employs three machines connected through the SJA1105 switch under test. Of the three machines, 2 generate traffic, while the third receives it. In practice, all three machines are in fact the LS1021A cores running OpenIL on each of the 3 boards.

The TSN switch under test is that of Board 2. The SJA1105 switches of Board 1 and Board 3 also forward traffic, but their configuration is fixed (not subject to change) for the entirety of the demonstration, and is “standard” (equivalent to a non-TSN-enabled L2 switch).

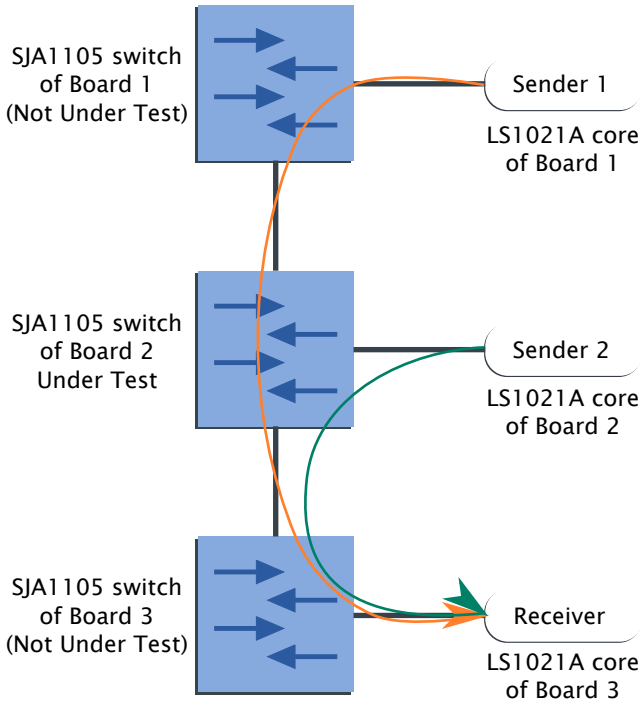


Figure 29. Connections of the three hosts and their roles as traffic senders/receivers

Through the SJA1105 switch of Board 2, there are two TCP flows competing for bandwidth:

- An iPerf3 connection running from client Board 1 to server Board 3
- An iPerf3 connection running from client Board 2 to server Board 3

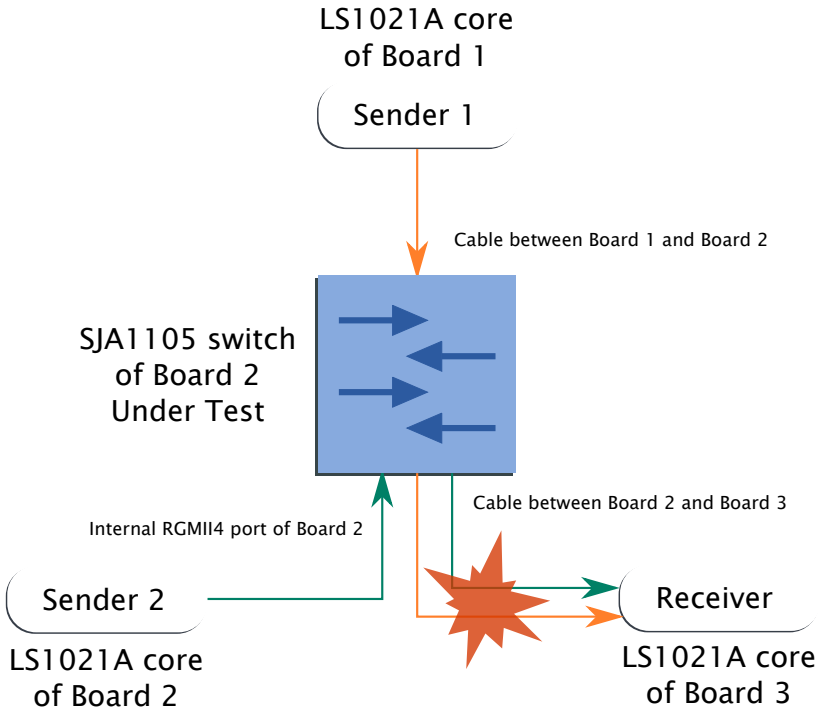


Figure 30. Simplified traffic flows for the rate limiting use case

Flows directed from Board 1 and Board 2 towards Host 3 are bottlenecked at the middle switch's egress interface The schematic diagram of the two iPerf3 flows is shown above. Flows directed from Board 1 and Board 2 towards Host 3 are bottle-necked at the middle switch's egress interface. The LS1021 on Board 3 is acting as iPerf server, whereas the ones on Board 1 and Board 2 are the iPerf clients. Since both these flows share the same link between the SJA1105 switches of Board 2 and Board 3, they are bottlenecked and compete for the 1000 Mbps total bandwidth of that link. The demo shows 3 approaches to isolate the flows' impact on one another (each of these approaches can be seen as an XML configuration applied to the SJA1105 switch of Board 2):

- Standard switch configuration: This is the behavior of traditional Ethernet switches.
- Ingress Policing: Rate-limit traffic coming from the LS1021 of Board 2 (in order to protect the flow Board 1 -> Board 3).
- Time Gating: Schedule the 2 flows on different time slots.

NOTE

In the following sections, unless otherwise specified, the term 'SJA1105' or just 'TSN switch' implicitly refers to the SJA1105 switch present on Board 2.

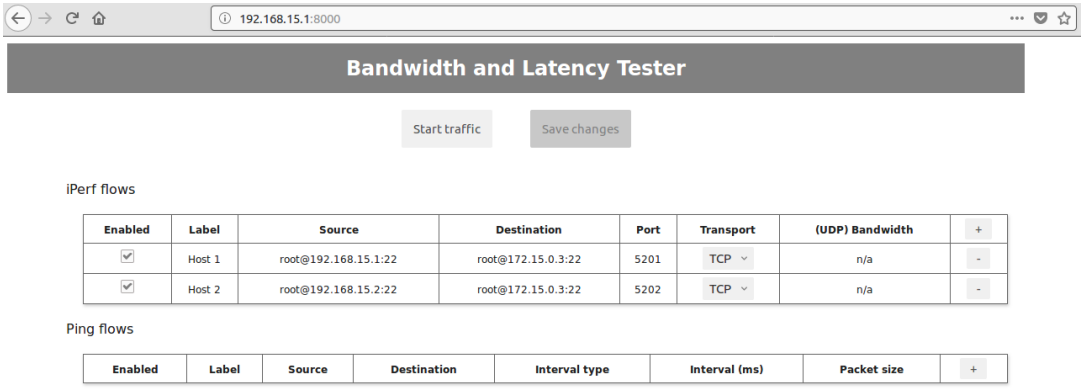
7.9.2 Objectives

- Manage bandwidth problems related to network contention
- Demonstrate the features of the L2 Ingress Policer
- Create time slots for scheduled traffic
- Show the usage of the sja1105-tool and helper scripts

7.9.3 Latency and bandwidth tester configuration

For this use case, two iPerf flows would be used as shown in the following figure.

Figure 31. iperf flows for bandwidth and latency tester



The preceding figure shows the LBT configured to generate two flows labeled "Host 1" and "Host 2", both destined to 172.15.0.3 (eth2 interface of Board 3), and originating from Board 1 and Board 2, respectively. It bears no importance whether the IP addresses of the iPerf sources are part of the management network (192.168.15.0/24) or TSN network (172.15.0.0/24). The route of the iPerf traffic is decided based on the requested destination address of the flow (in this situation, traffic goes through the TSN network).

7.9.4 Use of VLAN tags in the demo

The 802.1Q standard specifies that VLAN-encapsulated Ethernet frames have an additional 4 octet header with the following fields:

- **VLAN Ethertype**: must be set to 0x8100
- **VLAN Priority Code Point** (PCP)
- **Drop Eligibility Indication** (DEI)
- **VLAN ID**

In the second and third approaches of the demo (*Ingress Policing* and *Time Gating*), the SJA1105 must distinguish between the two flows, in order to prioritize them. To do so, it uses VLAN tags, specifically the PCP (priority) field.

The SJA1105 switch has three main stages in its packet processing pipeline:

- Ingress
- Forwarding
- Egress

On the ingress stage, the switch is configured to assign a default ("native") VLAN header on frames, based on their incoming port. Based on the default VLAN tagging, the flows receive differentiated treatment:

- In the policing configuration, one of the flows is rate-limited on the ingress port.
- In the scheduling configuration, similar rate-limiting effect is achieved as each flow gets its own time slot allocated for the forwarding and egress stages.

On the egress stage, the default VLAN tag is removed, so the connected hosts (Board 1, Board 2, Board 3) are oblivious to this VLAN tagging.

7.9.5 Standard configuration

Prepare the 3 boards with default L2 switch configurations:

- Ingress Policer is disabled on all ports
- All frames are internally tagged with a VLAN priority of 0 and are, as such, treated as equal when forwarded
- Qbv engine is not configured
- All ports are enabled for forwarding traffic

The XML configuration for this case was generated by running this sja1105-tool helper script:

Boards 1 and 3 make use of the default, built-in configuration of the sja1105-tool, while Board 2 loads it from standard.xml.

Board 1:

```
[root@board1] $ sja1105-tool config default ls1021atsn
[root@board1] $ sja1105-tool config upload
# Shorthand version:
# sja1105-tool config default -f ls1021atsn
```

Board 2:

```
[root@board2] $ sja1105-tool config load standard.xml
[root@board2] $ sja1105-tool config upload
# Shorthand version:
# sja1105-tool config load -f standard.xml
```

Board 3:

```
[root@board3] $ sja1105-tool config default ls1021atsn
[root@board3] $ sja1105-tool config upload
# Shorthand version:
# sja1105-tool config default -f ls1021atsn
```

Note that the configuration provided in standard.xml is equivalent to that of the built-in one. This can be seen by running:

Board 2:

```
# Load the built-in configuration into the staging area (no SPI write)
[root@board2] $ sja1105-tool config default ls1021atsn
# Export the configuration from the staging area to an XML file
[root@board2] $ sja1105-tool config save builtin.xml
# Compare the two
[root@board2] $ diff builtin.xml standard.xml
# No output means match
```

7.9.5.1 Ingress Policer

The L2 Ingress Policer inside the SJA1105 is implemented as a Token Bucket:

- Bucket max size (also known as burst size) is called SMAX (maximum is 0xFFFF)
- Bucket refill speed is RATE bytes per second (up to a maximum of 64000)
- Each ingress packet removes from the bucket a number of tokens equal to its length in bytes
- Can also police traffic based on maximum frame size

The Policing table has 45 entries:

- One for each Ingress Port x VLAN PRIO (5 x 8)
- One for Broadcast Traffic coming from each Ingress Port (5)

In the standard configuration, the L2 Ingress Policer is "**deactivated**". This means that RATE and SMAX are set to maximum (0xFFFF, 0xFA00) for all entries, so rate limiting can never occur at the maximum ingress rate of 1000Mbps.

This can be seen by looking at the l2-policing-table entries:

```
[root@board2] $ sja1105-tool conf show l2-pol
```

7.9.5.2 Default VLAN assignments

These are configurable through the MAC Configuration Table (5 entries, one per port are available through the SPI registers of the SJA1105 switch). Default VLAN tags are added only if the switch received the packets as untagged. The user can select whether the switch includes the VLAN tags in the egress packet or not. VLAN priorities are taken into consideration for the L2 Forwarding stage.

In the standard configuration, all ingress ports get by default VLAN priority 0 (best-effort) and all egress ports remove VLAN tags from packets.

7.9.5.3 Queuing diagram

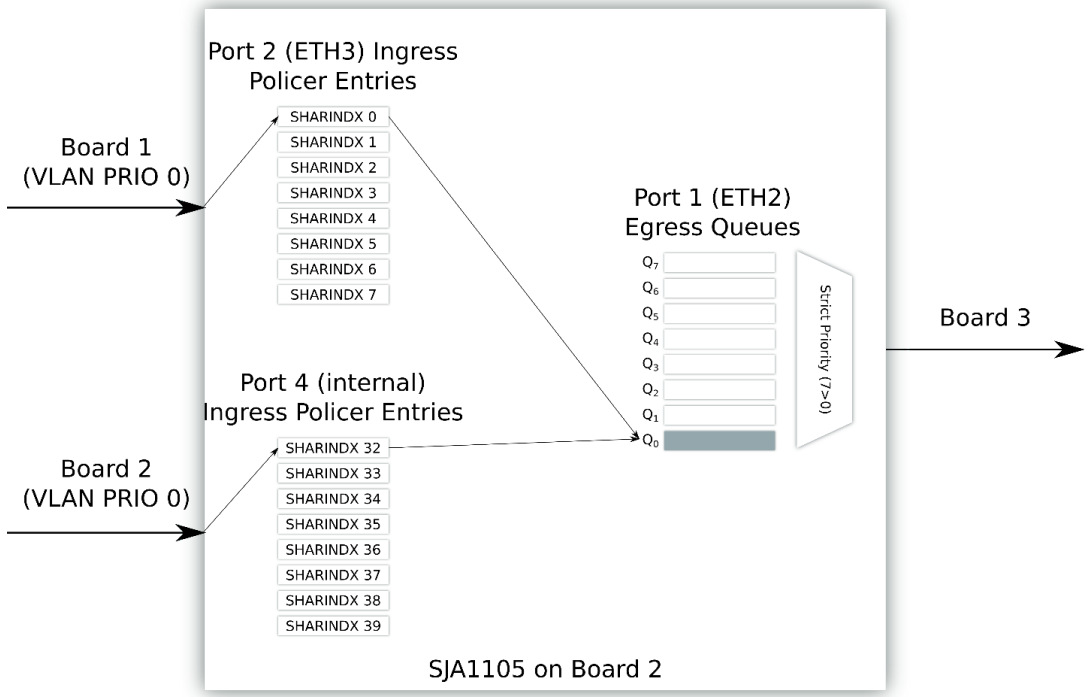


Figure 32. Queuing diagram

/>

The above figure shows how all traffic gets assigned to VLAN priority 0, causing contention on the same egress queue.

7.9.5.4 Results for the standard configuration

This section describes the results for different flows for the standard configuration.

In the LBT web app, run the following three tests:

- Only flow 1 enabled

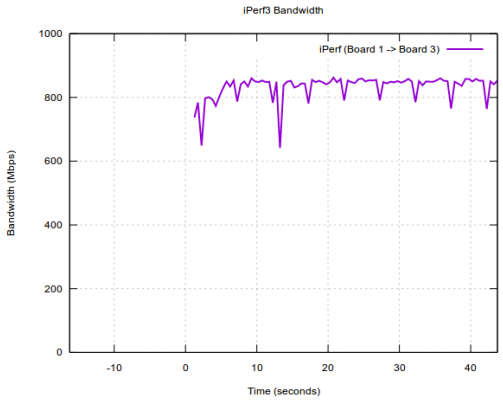


Figure 33. Standard configuration: Flow 1 run by its own

- Only flow 2 enabled

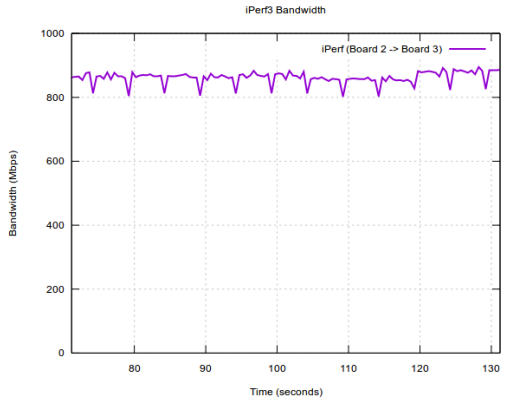


Figure 34. Standard configuration: Flow 2 run by its own

- Both flows enabled

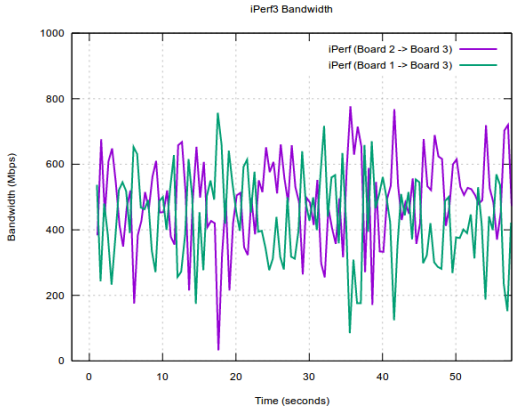


Figure 35. Standard configuration: Both flows run at the same time

Comments:

- Individually, both Board 1 and 2 get around 950 Mbps.
- Run at the same time, bandwidths for Flow 1 and Flow 2 oscillate.
- Bandwidth allocation is suboptimal (sum of the flows is much lower than 1000 Mbps).

7.9.6 Prioritizing configuration

In this case, configure the SJA1105 switch of Board 2 to assign these default VLAN priorities for untagged traffic:

- Board 1: VLAN PCP 5
- Board 2: VLAN PCP 3

This is done on a per-ingress port basis (Board 1 - Port RGMII0 (ETH3), Board 3 - Port 4 (internal)). This means that all untagged traffic received by the SJA1105 under test (Board 2) on the respective ports would have this VLAN tag appended for internal processing. Frames that are already VLAN tagged (not applicable to this scenario) are not altered.

On the egress port 2 (ETH3, towards Board 3), if flow 1's queue is not empty, the switch always prefers to send packets from that instead of flow 2's queue, because of its higher VLAN priority.

The XML configuration for this case was generated by running this sja1105-tool helper script:

```
[ubuntu@sja1105-tool/src/helpers] $ ./configs/rate-limiting/prioritizing.sh --flow1-prio 5 --
flow2-prio 3
Configuration saved as ./configs/rate-limiting/prioritizing.xml.
View with: "sja1105-tool config load ./configs/rate-limiting/prioritizing.xml; sja1105-tool
config show | less"
```

The SJA1105 configurations to use on Board 1 and Board 3 are the default, built-in ones that were programmed in the standard case.

7.9.6.1 Queuing diagram

The following figure shows the queuing diagram for the prioritizing configuration. It shows that Board 2 traffic is assigned a higher VLAN priority than Board 1. On egress, Board 2 dominates because of the strict priority queuing discipline of the switch.

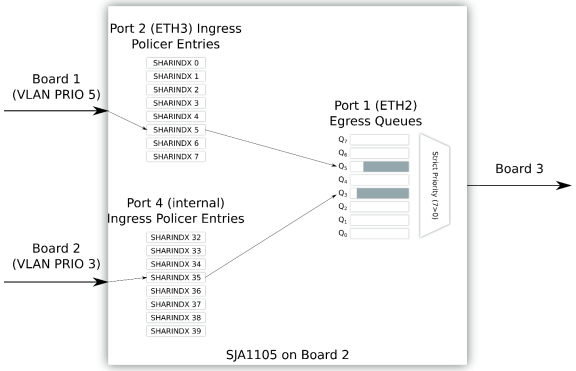


Figure 36. Prioritizing configuration

7.9.6.2 Results for the prioritizing configuration

This section describes the results for different flows for the prioritizing configuration.

In the LBT web app, run the following tests:

- Flow 1 run by its own

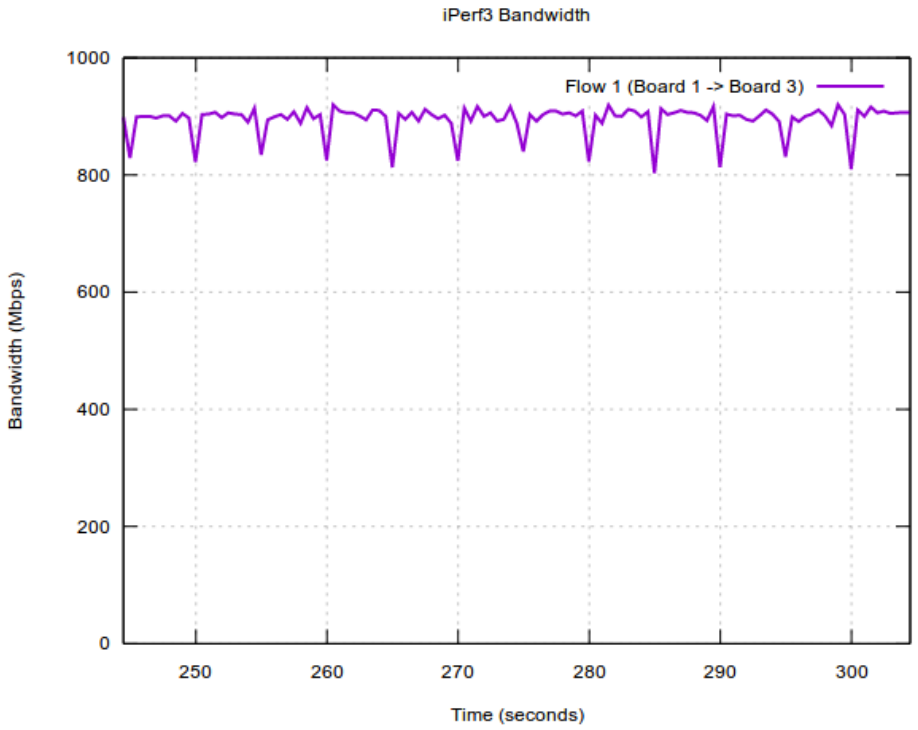


Figure 37. Prioritizing configuration: Flow 1 run by its own

- Flow 2 run by its own

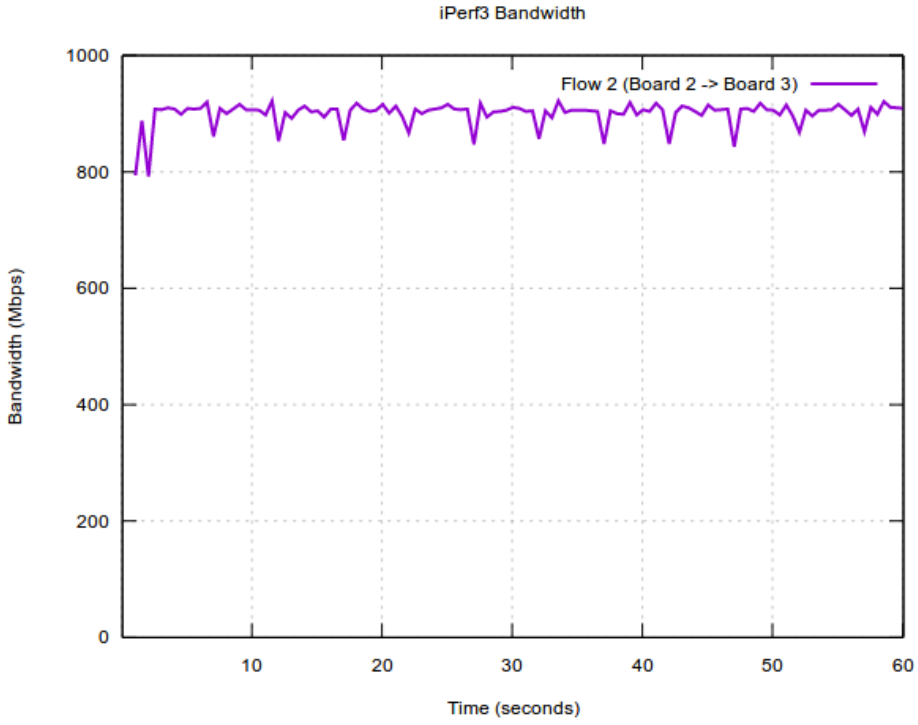


Figure 38. Prioritizing configuration: Flow 2 run by its own

- Both flows run at the same time

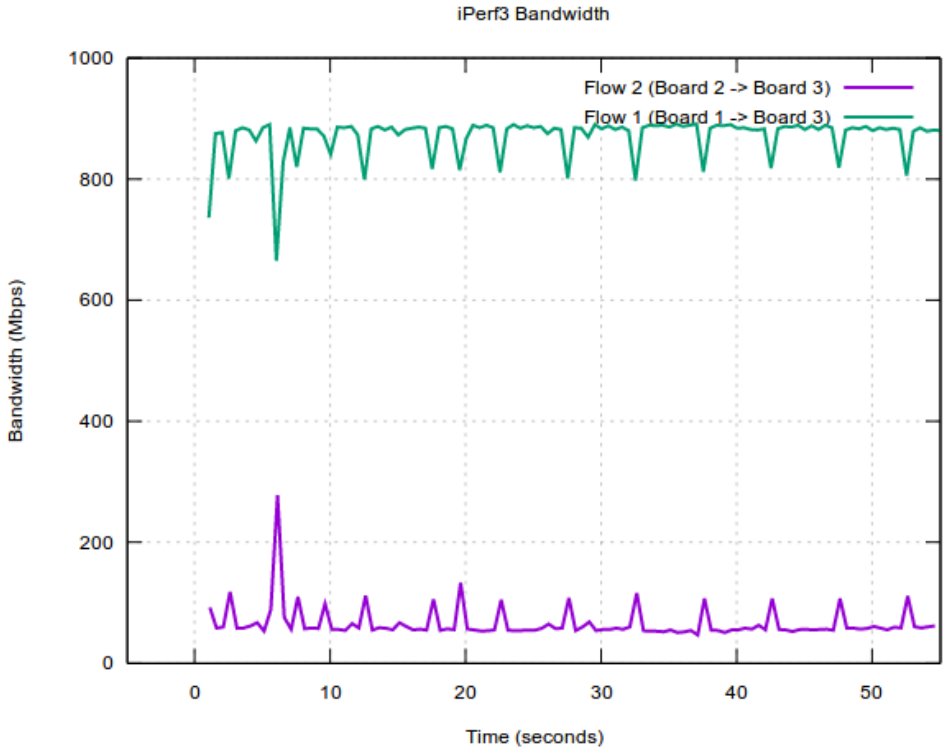


Figure 39. Prioritizing configuration: both flows run at the same time

Comments:

- Each flow, when run by its own, gets access to the full link bandwidth.
- When run at the same time, the switch applies strict priority between the flows, and Flow 1 (with a VLAN priority of 5) is protected, keeping the same bandwidth as when running alone.
- Flow 2 can only get the remaining bandwidth up to 1000 Mbps, which is typically very low.

7.9.7 Policing configuration

Based on the prioritizing configuration, we can apply rate limiting on Flow 1, since it has a higher VLAN priority and will obtain its rate-limited slice of the bandwidth, anyway.

The XML configuration for this case can be generated by running this sja1105-tool helper script:

```
[ubuntu@sja1105-tool/src/helpers] $ ./configs/rate-limiting/policing.sh --flow1-
prio 5 --flow2-prio 3 --flow1-rate-mbps 600
Configuration saved as ./configs/rate-limiting/policing.xml.
View with: "sja1105-tool config load ./configs/rate-limiting/policing.xml; sja1105-tool config show
| less"
```

7.9.7.1 Queuing diagram

The following figure shows the queuing diagram for the policing configuration. It shows the same queues as with the prioritizing configuration. The higher-priority traffic is rate-limited to allow the lower-priority traffic to use more of the remaining space.

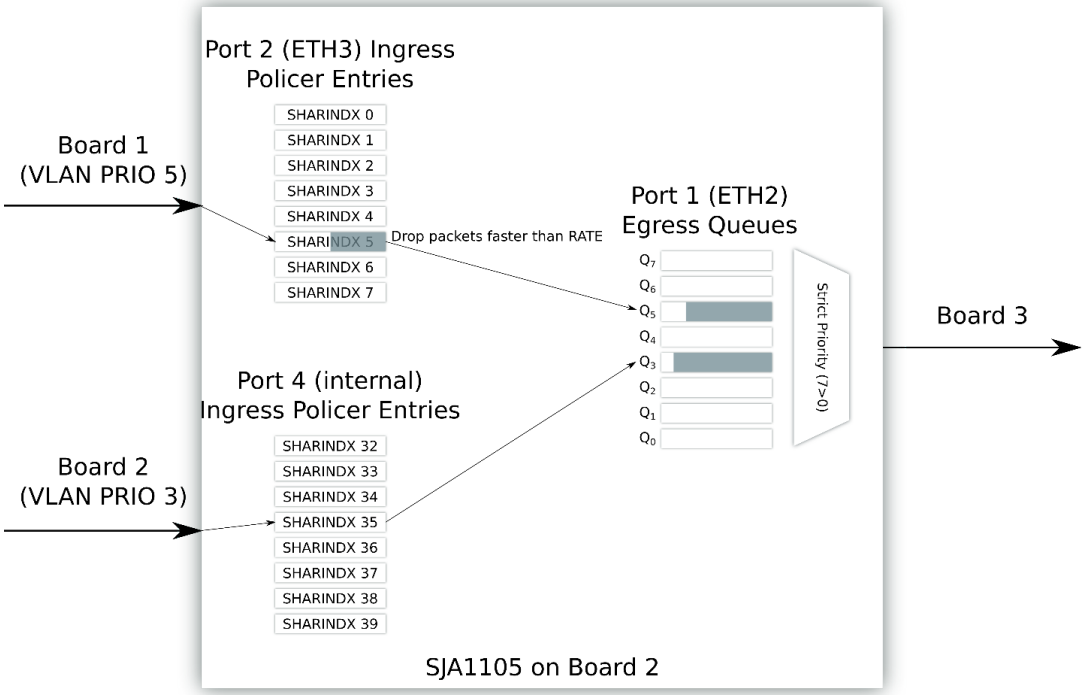


Figure 40. Queuing diagram for policing configuration

.

7.9.7.2 Results for the policing configuration

This section describes the results for different flows for the policing configuration. Run the following tests:

- Flow 1 run by its own

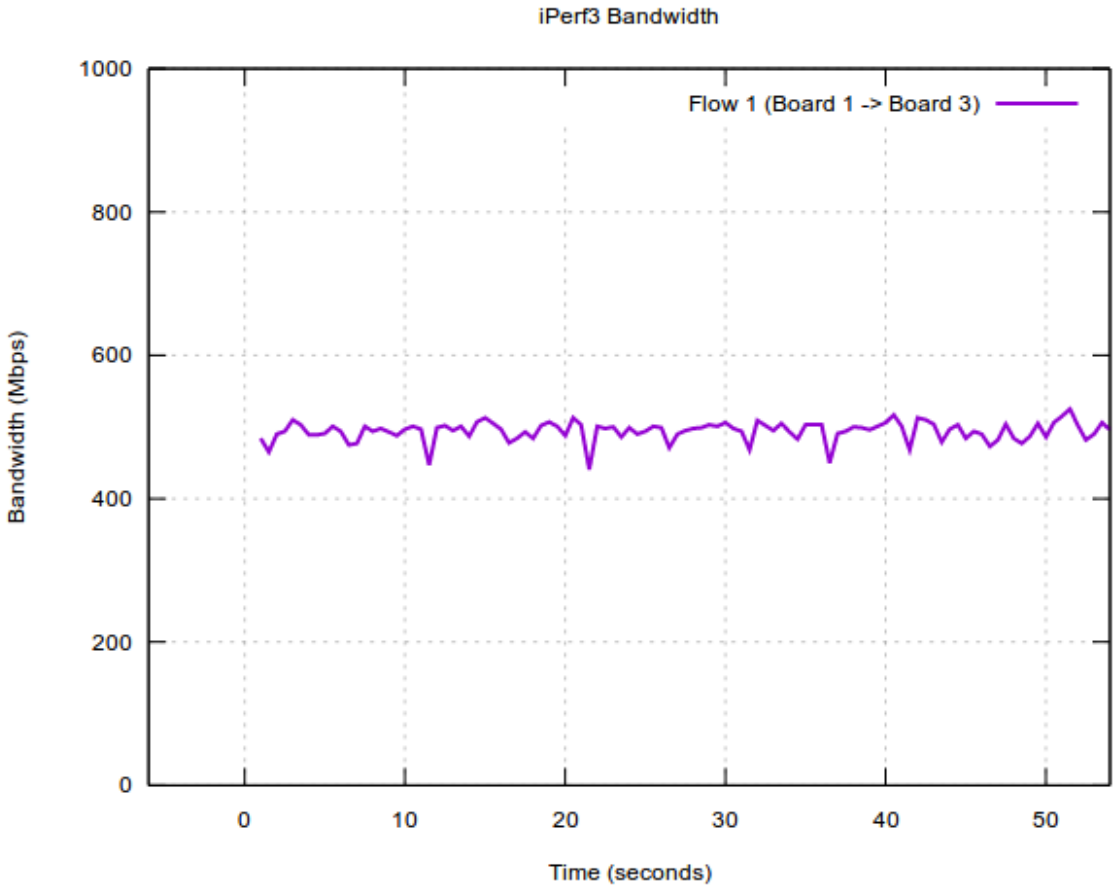


Figure 41. Policing configuration: Flow 1 run by its own

- Flow 2 run by its own

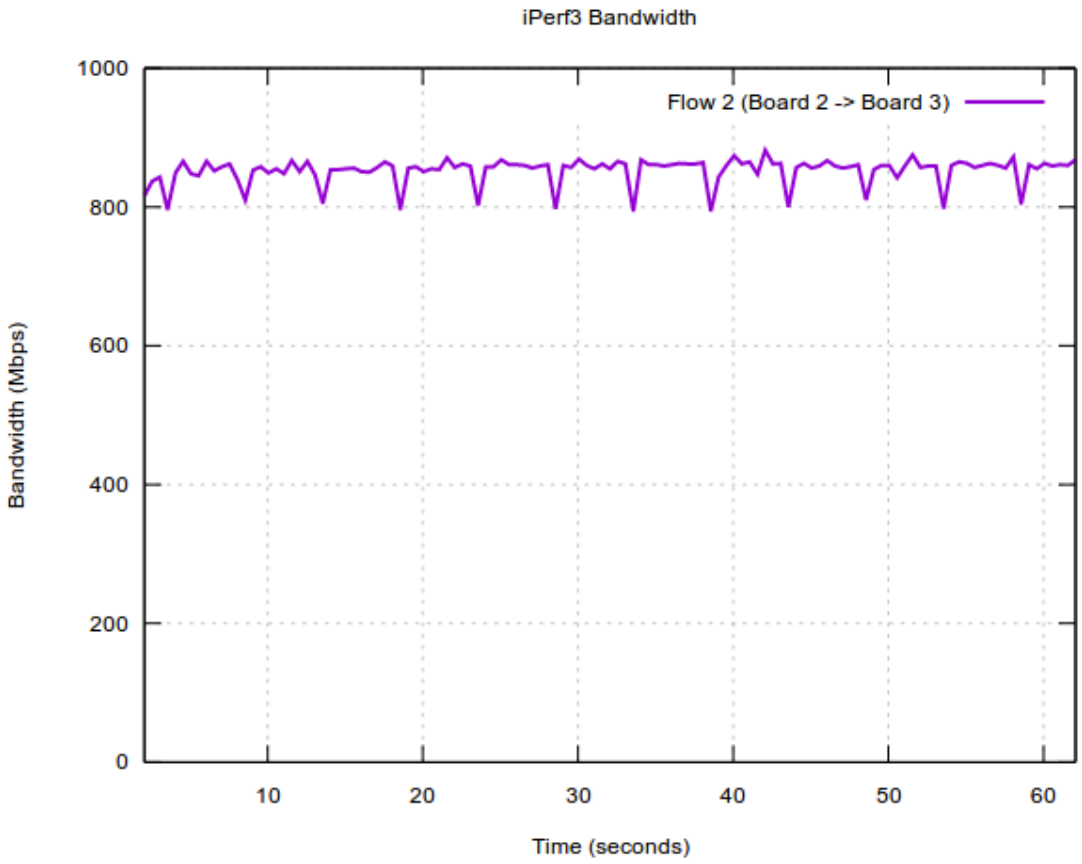


Figure 42. Policing configuration: Flow 2 run by its own

- Both flows run at the same time

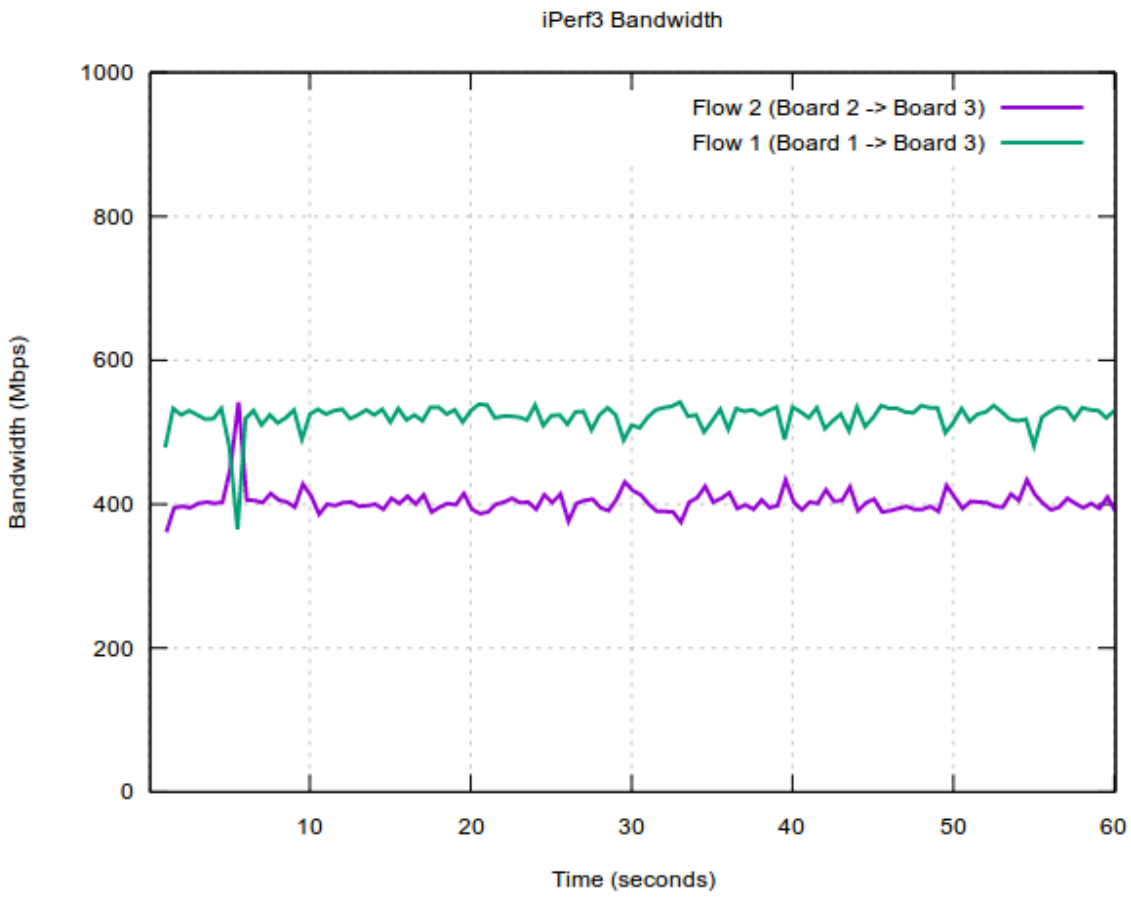


Figure 43. Policing configuration: both flows running at the same time

Comments:

- Using a combination of prioritization and policing, you can obtain the desired bandwidth allocation for both Flow 1 and Flow 2 (600-400).
- This is done by dropping part of the packets from Flow 1 (which might not always be desirable).
- In absence of the higher priority flow, Flow 2 is able to obtain line rate, because it is not rate-limited. The same cannot be said about Flow 1.

7.9.8 Scheduling configuration

The Time-Aware Scheduler of the SJA1105 switch works by following the guidelines in 802.1Qbv:

- Its 5 Ethernet ports each have 8 *gates* on egress, which can be open or closed
- Each *gate* controls its associated *queue* (there are 8 *queues*, one per traffic class priority)
- Whenever a *gate* is open, packets from its respective *queue* can be sent out the wire
- The Time Aware Scheduler (or Qbv engine) functions based on a clock ticking with a period of 200ns
- *Time slots* can be created, where some *gates* can be opened (allow certain traffic classes) and some can be closed. Each *time slot*'s action applies to some specified egress ports. A *time slot* has a defined period of time for which it is active, and is chained together with other *time slots* in a periodic *cycle*.

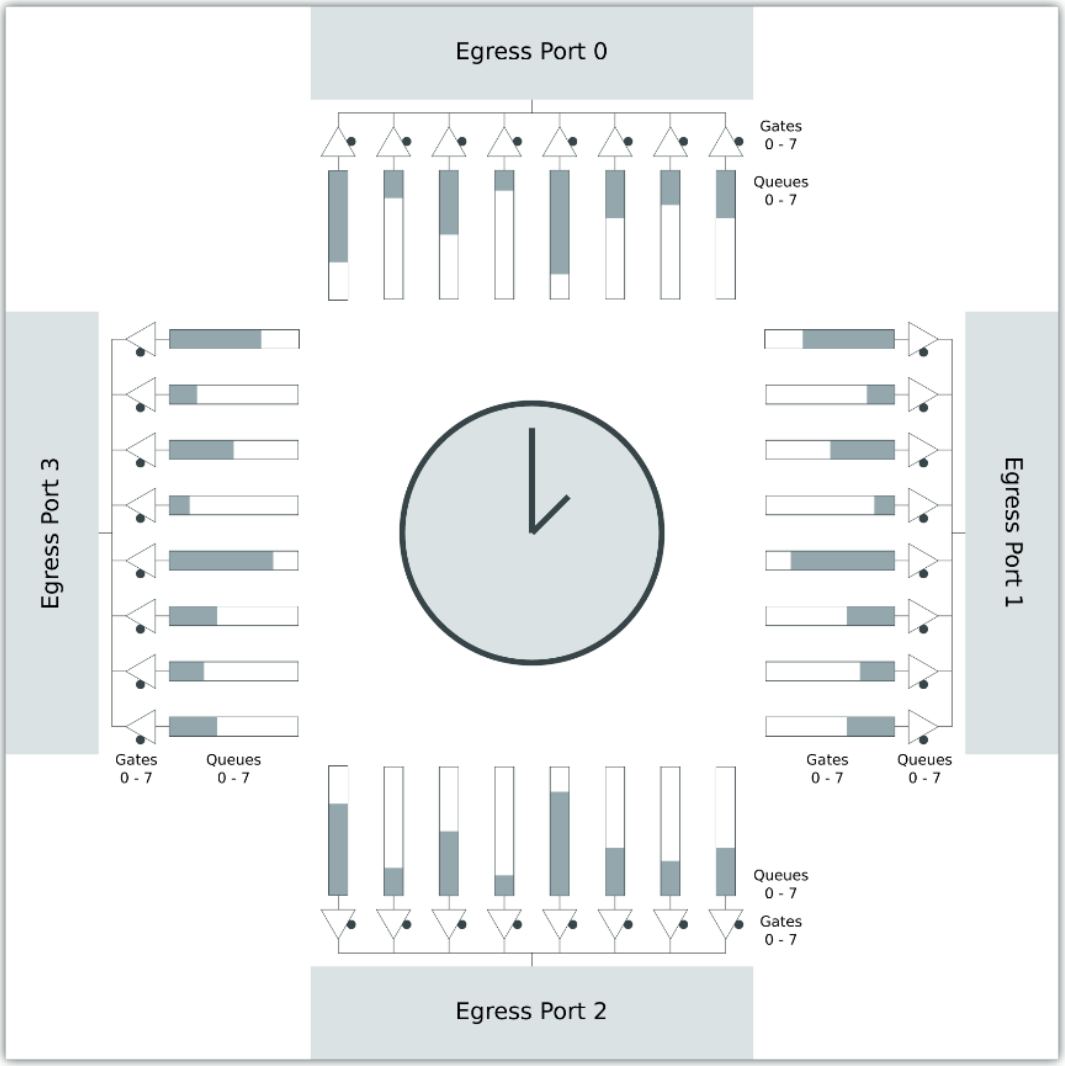


Figure 44. Structure of the Time Aware Scheduler

The user defines how many clock ticks each time slot (called *subschedule*, in SJA1105 terminology) takes, and also which flows (identified by their VLAN PRIO bits) are allowed to dequeue packets on each time slot. Once the Time-Aware Scheduler goes through each *time slot* (*subschedule*) in a round-robin fashion, it starts over again periodically. A complete period of subschedules is called a *schedule*.

In 802.1Qbv terminology, a *time slot* corresponds to a *SetGateStates* operation, and the length of the *cycle* is equal to *OperCycleTime*.

There are a total of 1024 entries allowed by the SJA1105 hardware for *time slots*. These can be grouped together to form at most 8 *cycles*, that run independently.

When defining multiple concurrent *cycles*, care must be taken to manually ensure that no two *time slots* ever trigger at the exact same moment in time.

In the third approach of the rate-limiting use case, the Time Aware Scheduler is active on the SJA1105 under test (that of Board 2) for egress port 1 (ETH2). This is the link towards Board 3, where the contention between Flow 1 and Flow 2 happens.

The XML configuration for this case was generated by running this sja1105-tool helper script:

```
[ubuntu@sja1105-tool/src/helpers] $ ./configs/rate-limiting/scheduling.sh --flow1-prio 5
--flow2-prio 3 --flow1-time-ms 6 --flow2-time-ms 4
```

```
Configuration saved as ./configs/rate-limiting/scheduling.xml.
View with: "sja1105-tool config load ./configs/rate-limiting/scheduling.xml;
 sja1105-tool config show | less"
```

The SJA1105 switch is configured to create a subschedule for VLAN PRIO 3 and one for PRIO 5. The total egress bandwidth is split 60% to Flow 1 (a time slot duration of 6 ms out of a total cycle length of 10 ms) and 40% to Flow 2.

In this configuration, Flow 1 is completely isolated from Flow 2, and there is minimal interference between the two, which allows better utilization of bandwidth.

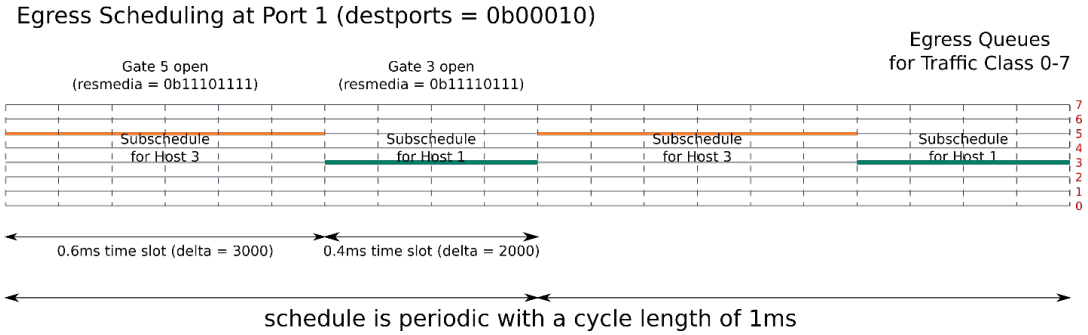


Figure 45. Time gating for Host 1 and Host 3, viewed on the time axis

There are four SJA1105 configuration tables that control the behavior of the Time-Aware Scheduler:

- Schedule Table
- Schedule Entry Points Table
- Schedule Parameters Table
- Schedule Entry Points Parameters Table

The Schedule Table contains the definitions of all the subschedules (time slots), or SetGateStates operations in Qbv terminology:

- What egress ports is the subschedule active on
- Which gates (egress queues for traffic classes) should be open and which should close
- The duration of the subschedule, in 200 ns increments

The Schedule Table does **NOT** define how the subschedules are linked together.

Each schedule has a starting point and an ending point, defined as indices to subschedules from the Schedule Table:

- The starting point is defined in the Schedule Entry Points table
- The ending point is defined in the Schedule Parameters table

For a more complete description of how the SJA1105 should be configured for Qbv operation, refer to the sja1105-tool helper script under `src/helpers/bin/scheduler-create`.

7.9.8.1 Results for the scheduling configuration

This section describes the results for ping testing for different flows for the Scheduling configuration.

- Flow 1 run by its own

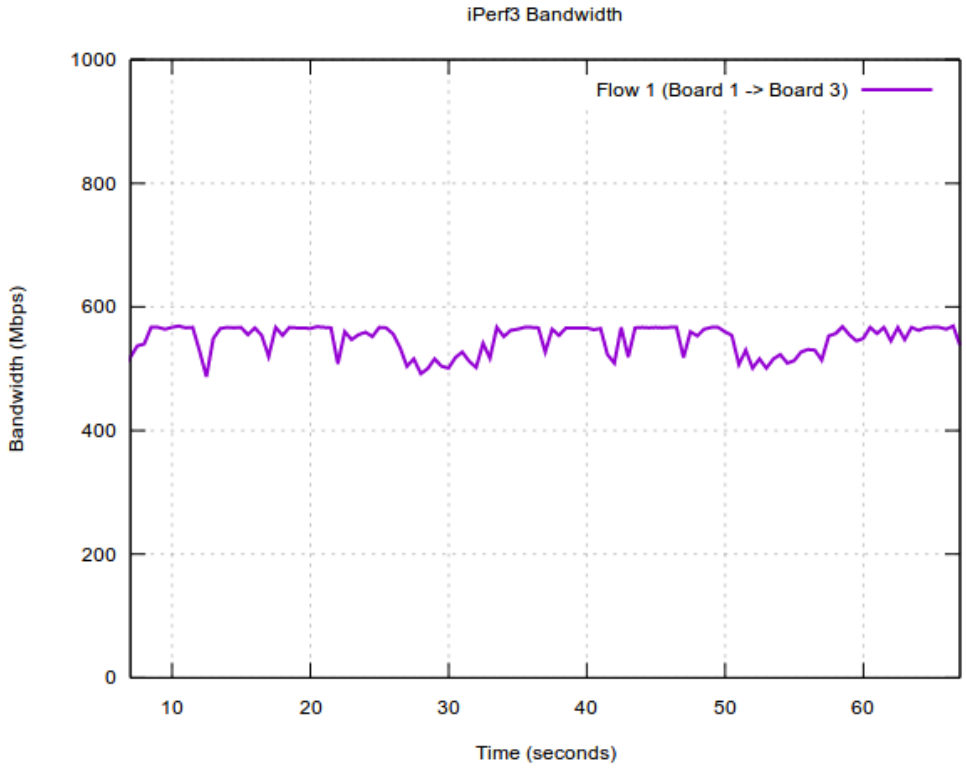


Figure 46. Scheduling configuration: Flow 1 run by its own

- Flow 2 run by its own

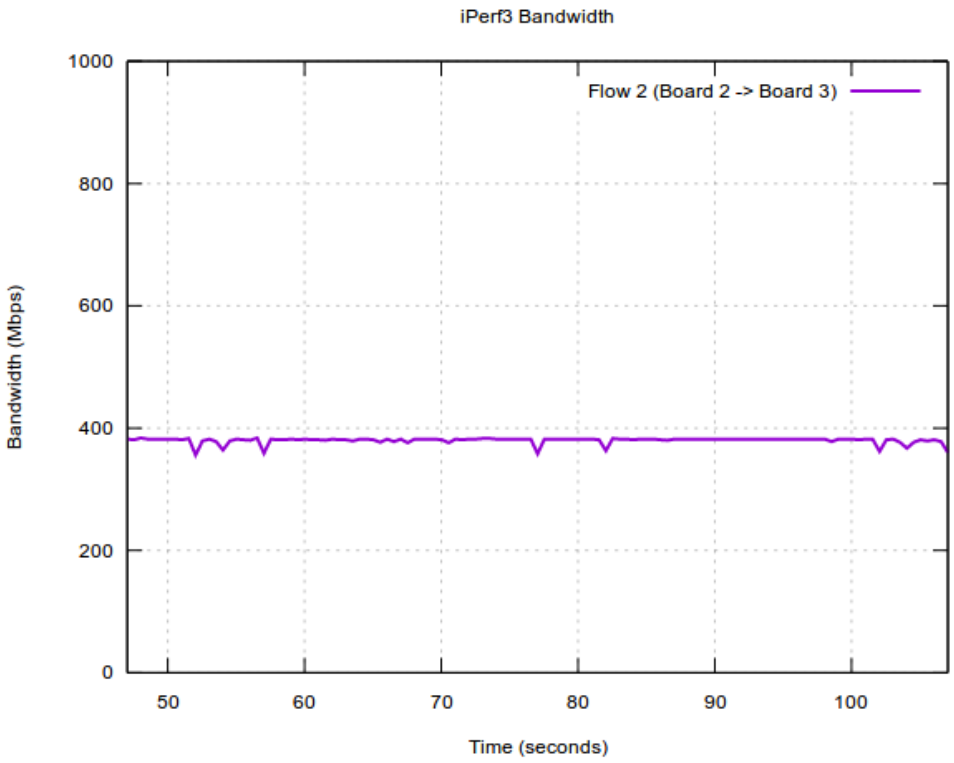


Figure 47. Scheduling configuration: Flow 2 run by its own

- Both flows run at the same time

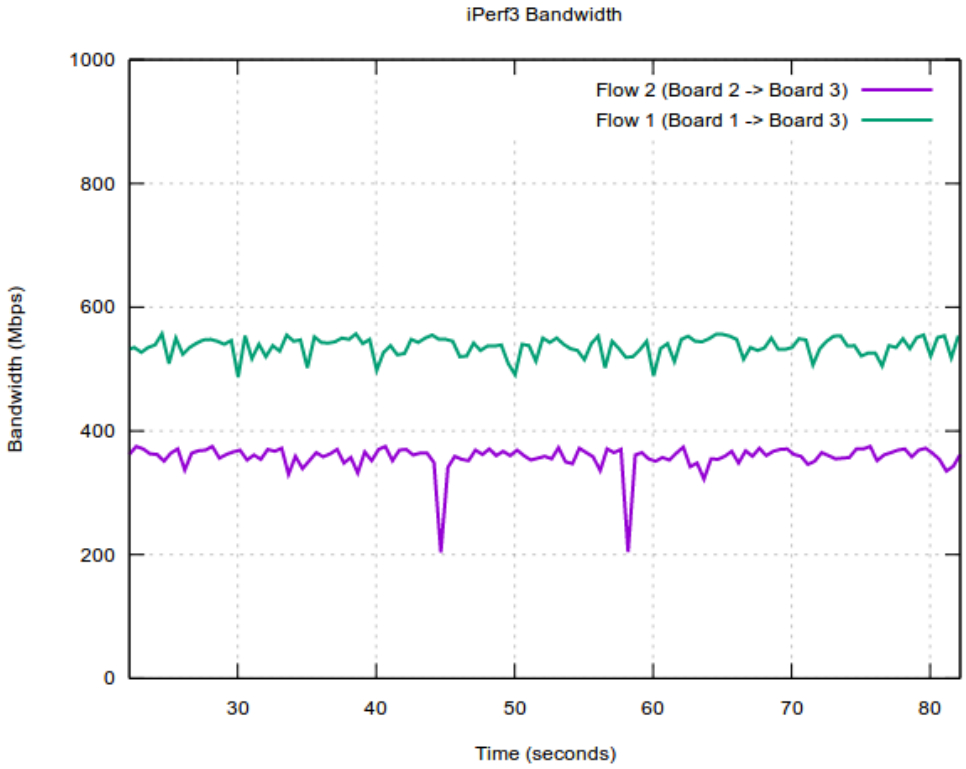


Figure 48. Both flows run at the same time

Comments:

- Regardless of being run separately or simultaneously, the two flows are allocated at 60% and 40% of the total Port 1 egress bandwidth by the Time-Aware Scheduler of SJA1105 on Board 2.
- The Time Aware Scheduler of the SJA1105 allows finer-grained control over bandwidth allocation.

7.9.9 Results of the demo

After running the steps described in the preceding sections, the following four XML files can be located in the OpenIL home directory of Board 2:

- standard.xml
- prioritizing.xml
- policing.xml
- scheduling.xml

The walkthrough must be followed step-by-step only once. Afterwards, a specific configuration can be loaded as shown in the commands below:

```
[root@openil] $ sja1105-tool config load standard.xml
[root@openil] $ sja1105-tool config upload
```

The SJA1105 configuration on the other boards must be kept default. Ensure these commands are run once, at the beginning of the tests:

```
[root@board1] $ sja1105-tool config default -f ls1021atsn
[root@board3] $ sja1105-tool config default -f ls1021atsn
```

7.10 Synchronized Qbv demo

This section describes the synchronized Qbv demo in brief, its objectives, Qbv schedule analysis, the various scenarios, setup preparation, and Latency and Bandwidth Tester (LBT) configuration.

7.10.1 Introduction

This demo covers a possible use case of the following TSN standards combined:

- IEEE 1588 (Precision Time Protocol)
- IEEE 802.1Qbv (Time-Aware Scheduling)

The scenario is to assure deterministic, fixed latency for a particular control flow in a switched Ethernet network, regardless of interfering traffic. Let's assume that Node A generates control events periodically, every 5 ms, and that these need to be propagated *as soon as possible* to Node B, which is situated 3 Ethernet switches away (3 hops) from Node A.

Trying to do this inside a regular, unconfigured Ethernet switched network usually results in latencies between Node A and Node B that invariably and uncontrollably increase as soon as there is any sort of background traffic through the network. This is because, by default, all frames are treated equally (best-effort) by the switches, which makes all traffic susceptible to unpredictable queuing delays.

The naïve solution to the queuing issue would be to simply raise the L2 priority of that specific control flow, such that Ethernet switches along its path always schedule that flow for transmission first.

The problems with this initial solution are twofold.

- Firstly, simply increasing a flow's priority puts too much trust in its well-behaving. If Node A malfunctions, or simply decides to send packets quicker than the 5 ms interval we accounted for, there is a high chance it will cause the other traffic flows, with less priority, to suffer from starvation.
- Secondly, although the control flow has the highest priority possible, it might still happen that it experiences forwarding delays. This is because by the time a control flow frame should be sent, the transmission medium might already be occupied transmitting a frame with less priority, which the switch must first finish sending. This is a form of priority inversion.

For the first issue, we can apply rate limiting on the control flow we just prioritized. This way, through prioritization we ensure a minimum guarantee of service, while through rate limiting we put a maximum limit to that guarantee.

Rate limiting can be applied in two ways: either with egress shaping on that specific port, or with ingress policing on the receiving side of that flow. Both these algorithms are generally implemented as a Token Bucket, which is a simple method to spread out heavy bursts of packets, either by dropping some (ingress policer) or delaying the sending of some (egress shaping).

The simple Token Bucket algorithm is only able to generate packets that are spread out evenly, given a specific time resolution. If more complex "waveforms" of packet transmission are desired, or if the timing accuracy must be very low, a different approach called Time-Aware Scheduling (TAS) can be employed.

The Time-Aware Scheduler, defined in **IEEE 802.1Qbv**, associates "gates" with each of the 8 priority queues connected to the egress ports. A gate is said to be "open" if frames are allowed to be selected for transmission from that gate's associated queue, or "closed" otherwise. A cyclic schedule is kept, where multiple timeslots define what is the state of every one of the gates (open or closed), and for how long.

Revisiting the question of how to minimize the impact of competing traffic flows on one another, one can configure the Time-Aware Scheduler with a single gate open per timeslot, effectively isolating the flows in time, and creating a Time-Division Multiple Access (TDMA) type of forwarding policy.

Even with the Time-Aware Scheduler, one issue still remains: the priority inversion caused by unfinished frame transmission at the end of its allocated timeslot. For this issue, two solutions exist: either IEEE 802.1Qbu (frame preemption), or allocation of an empty extra timeslot which serves as a guard band.

But even then, if there are multiple Time-Aware Schedulers in the same L2 network, they need to have a common notion of time. By synchronizing the clocks of all Time-Aware Schedulers using **IEEE 1588** (PTP), frames can be forwarded in a coordinated manner, similar to synchronized traffic lights.

With careful planning of the schedule, each packet always reaches the destination with the same predictable latency. The goal of the synchronized Qbv demo is for frames to spend almost no time being buffered on nodes internal to the TSN network, but instead only at the entry point. Once the time comes for a frame to be transmitted towards the TSN network, it passes with minimal delay through it.

7.10.2 Objectives

The objectives of the demo are the following:

- Synchronize the SJA1105 PTP clocks using IEEE 1588.
- Generate SJA1105 XML configurations offline (on host) using a simplified JSON description, and upload them to the 3 boards over SSH or NETCONF.
- Run the SJA1105 Time-Aware Scheduler (Qbv engine) based off the PTP clock.
- Create a 3-switch TSN network with deterministic latency.
- Use ping traffic to determine the degree of synchronization between boards.
- Use iPerf3 as source of interfering traffic and prove it does not alter the ping latency.

7.10.3 Qbv schedule analysis

The Qbv schedule common to all three boards comprises of six timeslots, numbered 0 to 5.

The JSON description can be found inside the `src/helpers/configs/synchronized-qbv/qbv-ptp.sh` helper script from the `sja1105-tool` source tree and is displayed here for reference:

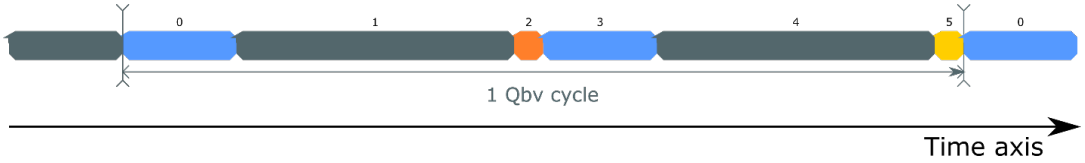
```
{
    "clksrc": "ptp",
    "cycles": [
        {
            "start-time-ms": "1",
            "timeslots": [
                {
                    "duration-ms": "4",
                    "ports": [${echo_port}, ${reply_port}],
                    "gates-open": [0, 1, 2, 3, 4, 5, 6],
    "comment": "regular traffic 1"
                },
                {
                    "duration-ms": "10",
                    "ports": [${echo_port}, ${reply_port}],
                    "gates-open": [],
                    "comment": "guard band 1"
                },
                {
                    "duration-ms": "1",
                    "ports": [${echo_port}],
                    "gates-open": [7],
                    "comment": "icmp echo request"
                },
                {
                    "duration-ms": "4",
                    "ports": [${echo_port}, ${reply_port}],
                    "gates-open": [0, 1, 2, 3, 4, 5, 6],
```

```
                    "comment": "regular traffic 2"
                },
                {
                    "duration-ms": "10",
                    "ports": [${echo_port}, ${reply_port}],
                    "gates-open": [],
                    "comment": "guard band 2"
                },
                {
                    "duration-ms": "1",
                    "ports": [${reply_port}],
                    "gates-open": [7],
                    "comment": "icmp echo response"
                }
            ]
        }
    ]
}
```

The echo_port and reply_port variables take individual values for each of the three boards.

The Qbv configuration can be summarized as follows:

Figure 49. Qbv schedule



- Ping traffic (ICMP echo request and echo reply) is classified by OpenIL through the `/etc/init.d/S45vlan` script and tagged with VLAN priority 7
- The blue time slots (0 and 3) represent regular traffic (traffic classes 0 through 6). The iPerf flows (traffic class 0) fall in this category.
- The grey time slots (1 and 4) represent periods of time where the switch allows no packet to be forwarded. These are guard bands for time slots 2 and 6.
- Time slot 2 (orange) allows ICMP Echo Request packets to be forwarded from source towards destination.
- Time slot 5 (yellow) allows ICMP Echo Reply packets to be forwarded from destination back towards source.

Correct functioning of the test (only a single ping packet will come through per Qbv cycle) is ensured by the fact that the ping is being run in "Adaptive" mode ("-A" flag).

The orange and yellow time slots must be large enough in order to counter potential time offsets between the three boards. This means that once a ping packet is forwarded by the first switch at the beginning of the time slot, there should be enough time left such that the same packet would also be forwarded in the same slot by the rest of the switches along the packet's path.

Although the orange and yellow time slots are long enough to permit forwarding multiple packets, in practice, at most two are forwarded per cycle (one ICMP echo request and one response) because ping is run in adaptive mode. Thus, there will be always be at most a single packet in flight, at any given moment.

- RTT (*Round-Trip Time*) is defined as the time interval between ICMP Echo Request i and ICMP Echo Reply i, both measured at the sender.
- PIT (*Packet Inter-arrival Time*) is defined as the time interval between ICMP Echo Request i and ICMP Echo Request (i +1), both measured at the receiver.

This use case analysis focuses on the Packet Inter-arrival Times measured at the receiver. This eliminates most delays caused by Linux user-space scheduling of the ping process and closely reflects the Qbv cycle length configured on the TSN switches.

The guard bands have two roles:

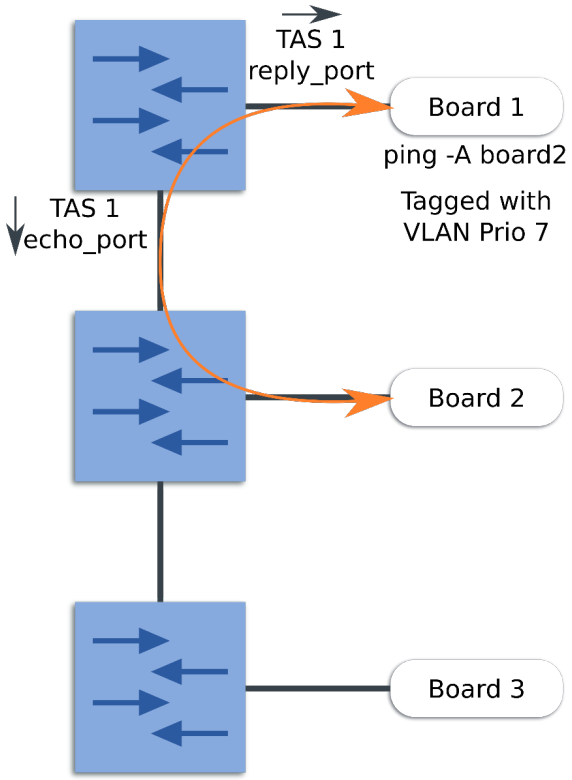
- Reduce the jitter while forwarding the ping packets through the network (ensure the switches have no other packets queued on the egress port when time slots 2 and 5 are scheduled for transmission. The guard bands are an alternative to IEEE 802.1Qbu Frame Preemption. A guard band duration of the time it takes to transmit an MTU-sized frame time at 1 Gbps is sufficient to eliminate this jitter.
- Let the LS1021 cores of the iPerf receiver cool down from receiving Rx interrupts from the network driver and finish processing the incoming traffic. This is important because, if upon receiving an ICMP echo request during time slot 2, the destination cannot process it and generates a reply until time slot 5 (15 ms), then a full Qbv cycle will be missed. The reported inter-arrival time in this scenario would be double (60 ms).

7.10.4 Scenarios

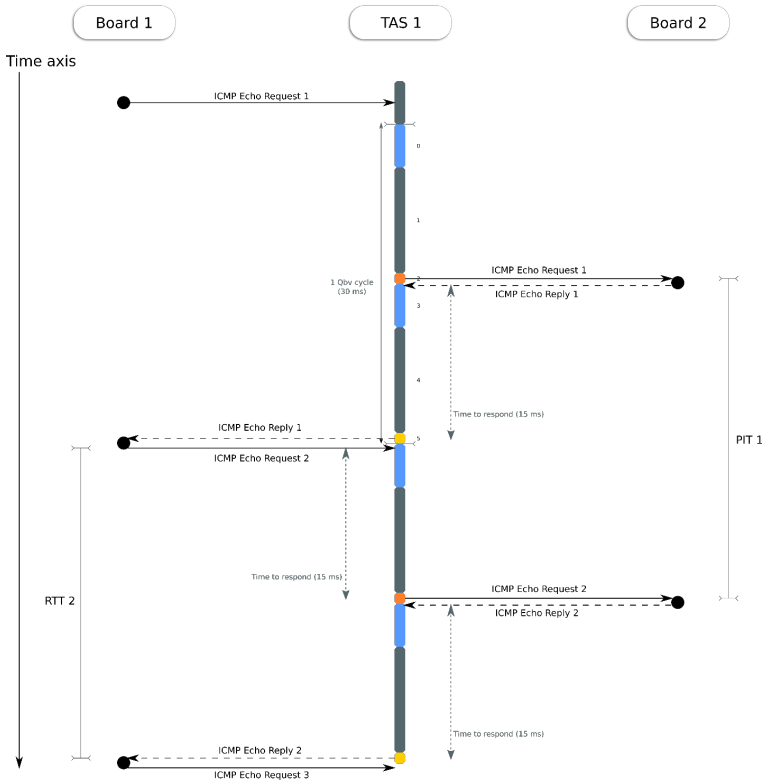
In both the 1-hop and 3-hop scenarios, adaptive ping is used to simulate the control flow packets sent from Node A to Node B. "Adaptive" ("-A" flag) means that the sending interval of Node A adapts to the RTT of the TSN network, which we are controlling directly by means of the Time-Aware Schedulers of the SJA1105 TSN switches along the path.

7.10.4.1 1 Hop scenario

In this scenario, Node A (ping sender) is Board 1, and Node B (ping receiver) is Board 2. Control traffic flows through a single Time-Aware Scheduler (TAS 1). The boards are connected as shown in the following figure.



A time visualization of ping packets in this 1-hop network looks as shown in the following figure.



On the time axis, on the left hand side are Round-Trip Times (RTT) reported by Node A (sender), while on the right-hand side are Packet Interarrival Times (PIT) reported by Node B (receiver).

The first RTT reported by Node A is expected to be random, since the ping sending interval is not yet aligned with the TAS cycle length. Afterwards, the forwarding of each subsequent ICMP Echo Request is expected to be delayed a full cycle by TAS 1, until it reaches Node B.

The Qbv scheduler on TSN switch 1 operates on two ports:

- On the `echo port`, during the orange time slot 2, where ICMP Echo Request packets are forwarded.
- On the `response port`, during the yellow time slot 5, where ICMP Echo Response packets are forwarded.

These time slots for ICMP traffic consume 2 ms out of the total Qbv cycle length of 30 ms (including their associated guard bands, the total rises to 22 ms out of 30 ms). In the rest of the Qbv cycle (8 ms, time slots 0 and 3), regular traffic (iPerf) is scheduled on both the echo and response port.

7.10.4.2 3-hop scenario

In this scenario, Node A (ping sender) is the Board 1 and Node B (ping receiver) is the Board 3. Control traffic flows through three Time-Aware Schedulers (TAS 1, TAS 2, and TAS 3). The boards are connected as shown in the following figure.

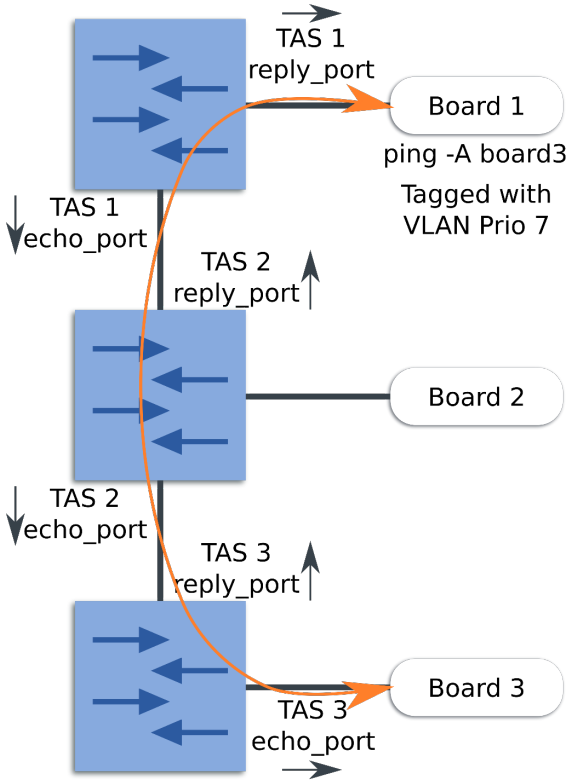


Figure 50. 3-hop scenario

The ideal scenario is when forwarding a ping packet takes a single cycle through all three hops. A time visualization of this scenario looks as shown in the following figure.

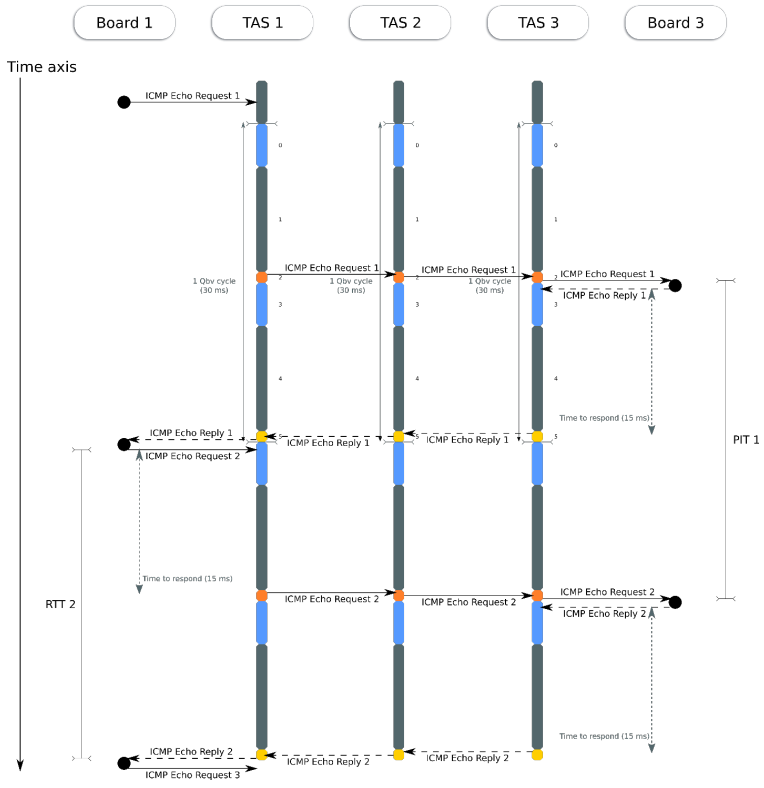


Figure 51. Timeslots in the 3 hop scenario

This scenario poses more difficulties, because the clocks of TAS 1, TAS 2, and TAS 3 must be in sync with one another. The expectation for this test is to get the same 30 ms interval reported as in the 1-hop case. This proves that using synchronized Qbv in a multi-switch network does not incur additional delays and can be used to ensure deterministic latency regardless of the number of hops.

As is the case with the 1-hop setup, TAS 1 delays the forwarding of ICMP Echo Requests until time slot 2 (orange). The key is that once TAS 1 forwards the ping packet, it is caught immediately (in the same cycle) by the orange time slot of TAS 2, and then by the green time slot of TAS 3.

As mentioned, this is possible because the lengths of the orange time slots are large enough to make up for potential PTP synchronization offsets between the boards.

The goal for the 3-hop scenario is for 100% of the ping packets to report an inter-arrival time (PIT) of 1 cycle (30 ms) at the destination, same as if it were a single hop.

7.10.5 Setup preparation

1. Inside a GNU/Linux environment, go to the SJA1105 helper scripts folder:

```
cd sja1105-tool/src/helpers
source envsetup
```

2. Generate the XML configurations for the 3 boards and upload them to the boards:

```
for i in 1 2 3; do ./configs/synchronized-qbv/qbv-ptp.sh --board ${i}; scp ./configs/
synchronized-qbv/qbv-ptp-board${i}.xml root@192.168.15.${i}:.; done
```

3. Open an SSH connection to Board 1.

4. Run the “tmux” command inside board 1’s terminal. Tmux is a “terminal multiplexer” that allows you to have multiple shells over the same SSH connection. Inside tmux, press the following keys: “Ctrl-a” and then “c”. Do this twice. You should now have 3 shells spawned inside tmux, as can be seen in the status bar at the bottom: “1:sh 2:sh 3:sh”:

5. Navigate to the first tmux shell by clicking on “1-sh”. Inside this shell, **load the XML configuration** into the sja1105-tool:

```
>sja1105-tool config load -f qbv-ptp-board1.xml
```

If successful, no output should be seen from this command.

6. Navigate to the second tmux shell by clicking on “2-sh”. Inside this shell, start the **LBT web application**:

```
/usr/lib/node_modules/lbt/server.js
```

7. As these XML configurations for SJA1105 use the PTP clock source for Qbv, you must run the PTP synchronization daemon inside the third tmux shell (3-sh):

```
ptp4l -i eth0 -p /dev/ptp0 -m -l 7 -t 1000
```

After a while where only this debug message is printed:

```
ptp4l[0000.000]: sja1105: sync timer timeout
```

The PTP daemon will begin to keep in sync the SJA1105 PTP clock with the eTSEC clock of the LS1021 eth0 port.

Synchronization offsets can be followed by examining these output lines:

```
ptp4l[46335.657]: sja1105: offset 202 ns, delay 94627 ns
```

The PTP daemon will also monitor and control the Qbv engine, which piggybacks its clock source from the PTP clock. The Qbv engine can be in one of 3 states:

- Disabled
- Enabled but not running (scheduled to begin in 3 seconds)
- Running

These states can be seen by examining the following output lines:

```
ptp4l[46335.658]: sja1105_qbv_monitor: state disabled
ptp4l[46335.916]: sja1105_qbv_monitor: state enabled not running
```

```
ptp4l[46335.917]: time to start: [2.870531024]
ptp4l[46702.166]: sja1105_qbv_monitor: state running
```

Under normal operation, the Qbv engine is expected to remain in the running state. Large PTP synchronization offsets will reset the PTP clock, synchronization algorithm, and thus, also the Qbv state machine.

Note that while the Qbv engine is not in the running state (either disabled or scheduled to begin), ping traffic is forwarded freely, and not rate-limited or protected.

8. Open an SSH connection to Board 2, open tmux and create 3 shells.
9. In the first shell (“1:sh”), load the XML configuration into sja1105-tool:

```
sja1105-tool config load -f qbv-ptp-board2.xml
```

10. In the second shell (“2:sh”), start the OPC UA server:

```
opc-sja1105
```

11. In the third shell, start the PTP synchronization daemon:

```
ptp4l -i eth0 -p /dev/ptp0 -m -l 7 -t 1000
```

12. Open an SSH connection to Board 3; open tmux and create 2 shells.
13. In the first shell (“1:sh”), load the XML configuration into sja1105-tool:

```
sja1105-tool config load -f qbv-ptp-board3.xml
```

14. In the second shell, start the PTP synchronization daemon:

```
ptp4l -i eth0 -p /dev/ptp0 -m -l 7 -t 1000
```

7.10.6 Latency and Bandwidth Tester Configuration

On Board 1, ensure that the file `/usr/lib/node_modules/lbt/config.json` has the following content:

```
{
    "listenPort": 8000,
    "sshPrivateKey": "/etc/ssh/ssh_host_rsa_key",
    "ping": {
        "plotStyle": "lineGraph",
        "xlabel": "Time (seconds)",
        "ylabel": "PIT (ms)",
        "xmin": "0",
        "ymin": "0",
        "xlen": "60",
        "binwidth": "0.01",
        "measurement": "pit",
        "measurementInterface": "eth2",
        "title": "Ping Packet Inter-arrival Time"
    },
    "iperf": {
        "plotStyle": "lineGraph",
        "xmin": "0",
        "ymin": "0",
        "xlen": "60",
        "xlabel": "Time (seconds)",
        "ylabel": "Bandwidth (Mbps)",
        "title": "iPerf3 Bandwidth"
```

```
        }
    }
```

If necessary, restart the LBT server after updating its configuration file.

In the browser, navigate to http://192.168.15.1:8000.

Input the following flows, as shown in the following figure:

- **Flow 1**: iPerf from Board 1 to Board 3 (over untagged TSN network)
- **Flow 2**: iPerf from Board 2 to Board 3 (over untagged TSN network)
- **1 Hop**: Adaptive Ping from Board 1 to Board 2 (over VLAN-tagged TSN network)
- **3 Hops**: Adaptive Ping from Board 1 to Board 3 (over VLAN-tagged TSN network)

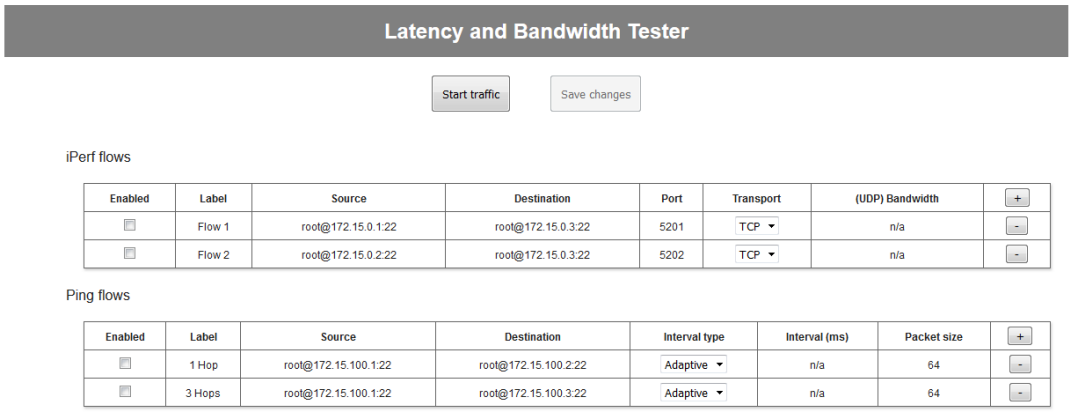


Figure 52. LBT configuration of flows for the Synchronized Qbv demo

7.10.7 Ping testing

The purpose of the ping test is to verify that the Qbv schedule engine is properly configured and synchronized across all three TSN boards, regardless of the number of hops and the background traffic.

7.10.7.1 1-hop flow

On the LBT web page, enable only the 1 Hop flow (from 172.15.100.1 to 172.15.100.2).

This passes through the TAS of Board 1.

Expected behavior: In this case, a constant 30 ms PIT measured on Board 2, can be observed, as shown in the following figure.

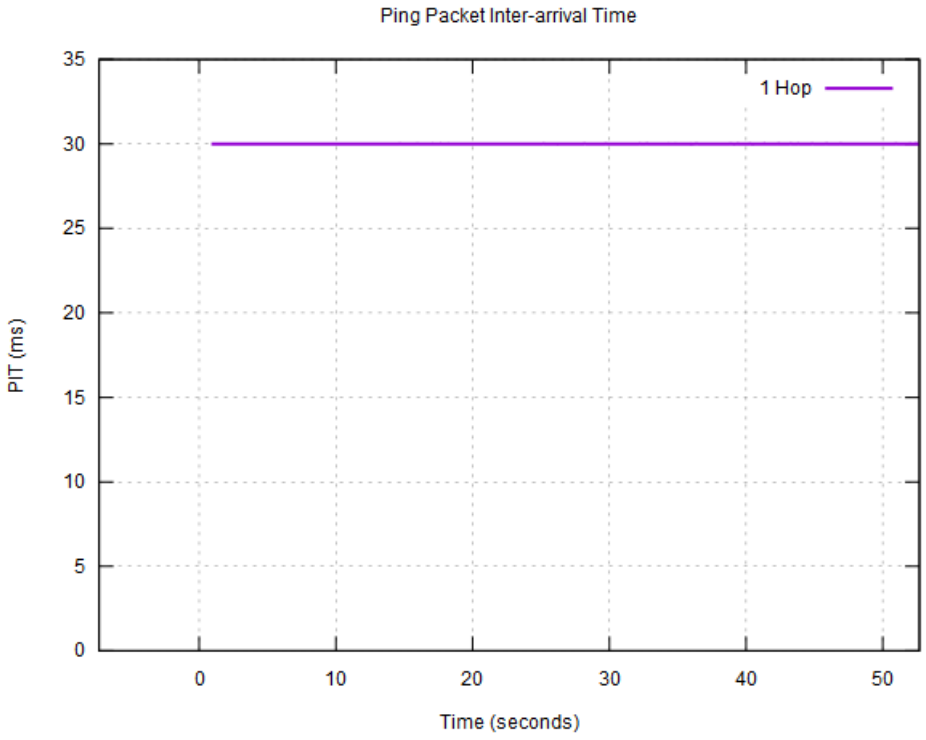


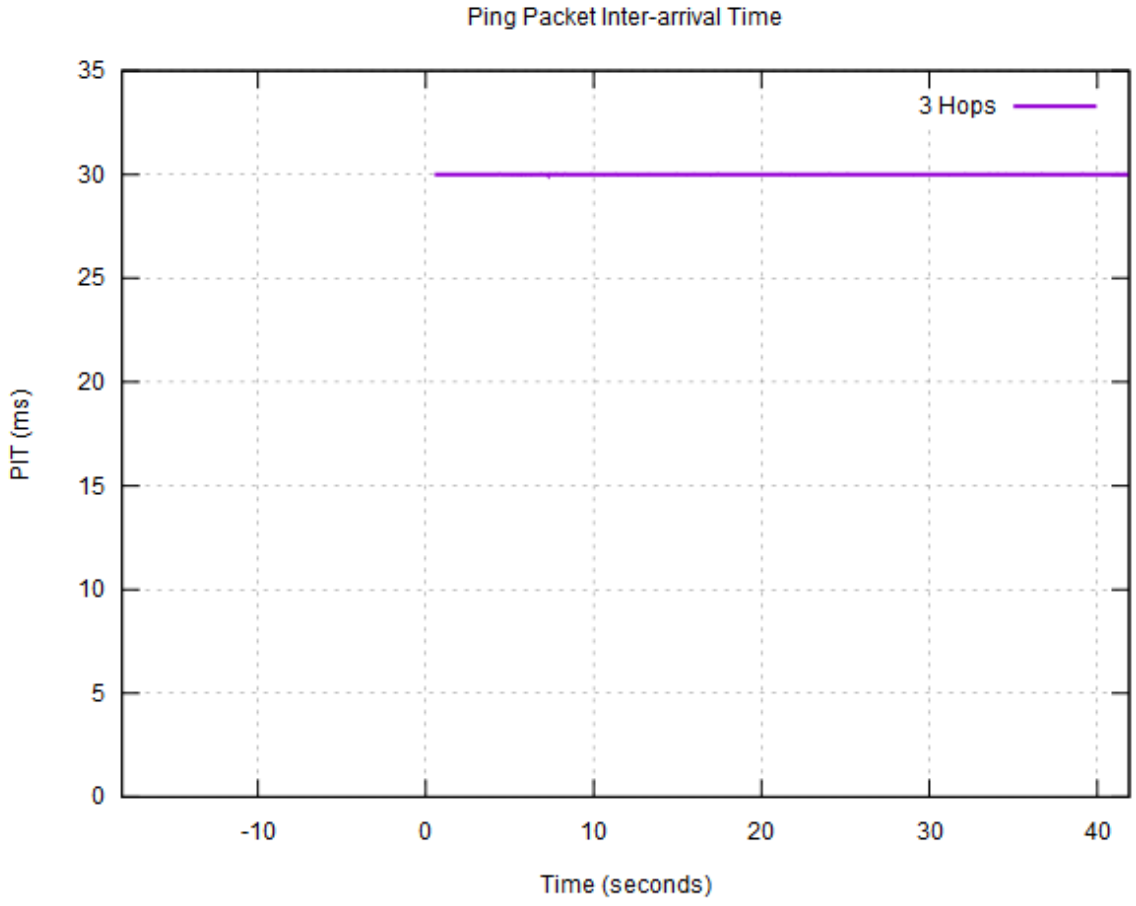
Figure 53. 1-hop flow

7.10.7.2 3-hops flow

Enable only the 3 Hops flow on the LBT web page. This flow passes through the TAS of Board 1, Board 2, and Board 3.

Expected behavior: You can observe the same 30 ms PIT, despite having added two extra hops in the TSN network as shown in the following figure.

Figure 54. 3-hops flow



7.10.7.3 1-hop with background traffic

Enable the following flows on the LBT web page:

- Flow 1 (iPerf)
- Flow 2 (iPerf)
- 1 Hop (Ping)

Expected behavior: The ping traffic is protected by TSN switch 1 from interference with the iPerf and retains a packet inter-arrival time of 30 ms at Board 2. This is shown in the following figure.

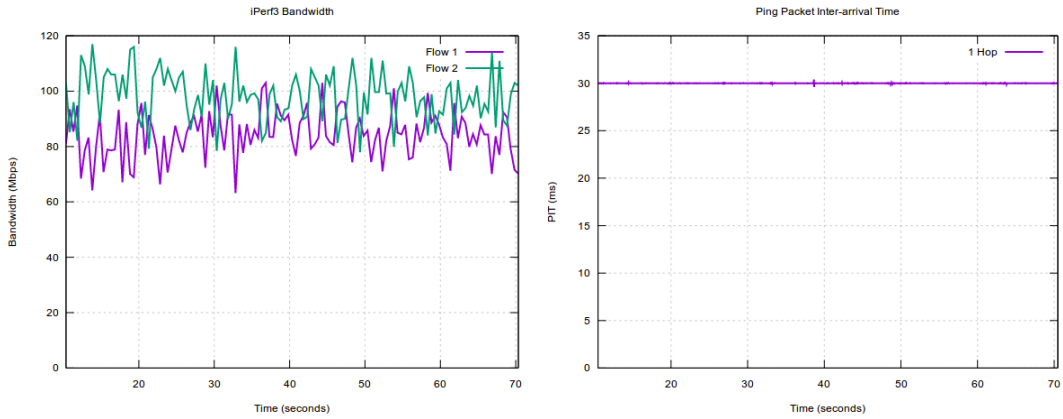


Figure 55. 1-hop ping with background traffic

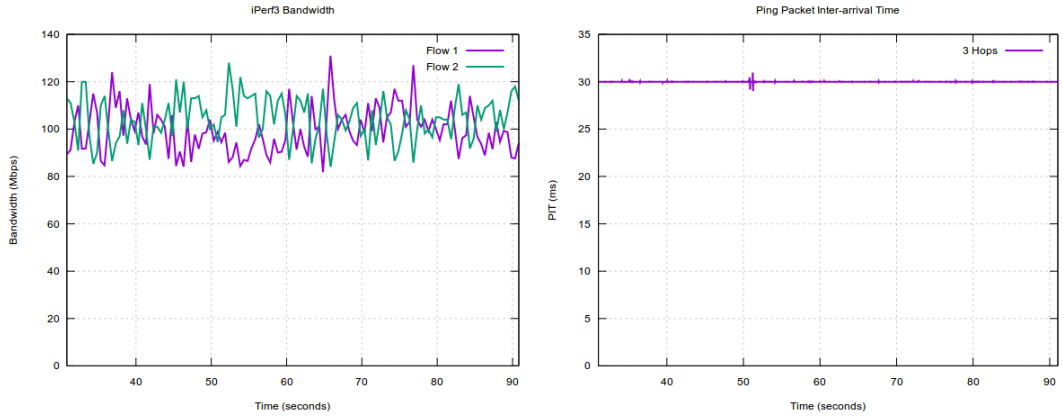
7.10.7.4 3 Hops with background traffic

Enable the following flows on the LBT web page:

- Flow 1 (iPerf)
- Flow 2 (iPerf)
- 3 Hops (Ping)

Expected behavior: the ping traffic is protected by TSN switch 1 from interference with the iPerf and retains a packet inter-arrival time of 30 ms at Board 2.

Figure 56. 3-hop ping with background traffic



7.11 NETCONF usage

Examples are provided only for Board 1 (IP 192.168.15.1). You need to repeat all steps described in this section for Board 2 and Board 3.

7.11.1 Creating a NETCONF session

Create three connections open to all three boards, each in a separate window using the following set of commands.

```
[ubuntu] $ netopeer-cli
netconf> connect --port 830 --login root 192.168.15.<board{1|2|3}-ip>
netconf> # press Enter for no password
# You may need to run this, in case you are using the candidate
# datastore (here we are not) and it becomes locked.
# See 5.6.5.
# netconf> discard-changes
```

7.11.2 Applying the configuration over NETCONF

For applying the configuration over NETCONF, repeat the following commands for board1, board2, and board3:

```
# Apply qbv-ptp-board1.xml to the running datastore
# Configuration takes effect immediately
netconf> edit-config --config qbv-ptp-board1.xml running
# Inspect the running datastore
netconf> get-config running
```

7.11.3 Running a configuration at startup

Repeat the following commands for Board1, Board2, and Board3:

```
netconf> copy-config --source running startup
```

7.11.4 Loading an existing XML configuration into the NETCONF datastore

After running the synchronized Qbv demo steps, on each board there should be a configuration file named /root/qbv-ptp-board{1,2,3}.xml.

The NETCONF server running on the board can be instructed to load it into its datastore:

netconf> user-rpc

<!--#

Type the content of a RPC operation.

-->

```
netconf> user-rpc
  <!--#
    Type the content of a RPC operation.
   -->
<load-local-config xmlns="http://nxp.com/ns/yang/tsn/sja1105">
    <configfile>
       board1-qbv-sync.xml
    </configfile>
</load-local-config>
```

Running the preceding command also applies the configuration.

7.11.5 Transferring the SJA1105 configuration to Ubuntu

It is also possible to retrieve and inspect the configuration from the board using NETCONF and netopeer-cli commands:

```
netconf> get-config running --out qbv-ptp-board1-retrieved.xml
```

After successfull completion of this operation, a new file named `qbv-ptp-board1-retrieved.xml`, is created in the current working directory on Ubuntu, with the current contents of the datastore of the netopeer-server that we are connected to. Assuming you followed along over step 0, this should match exactly the content of `qbv-ptp-board1.xml` on Board 1.

Proceed and transfer the contents of all XML configurations to the Ubuntu PC.

At the end of this step, the current working directory should have the following files:

- `qbv-ptp-board1.xml`
- `qbv-ptp-board2.xml`
- `qbv-ptp-board3.xml`
- `qbv-ptp-board1-retrieved.xml`
- `qbv-ptp-board2-retrieved.xml`
- `qbv-ptp-board3-retrieved.xml`

7.11.6 Viewing port statistics counters

The NETCONF protocol (and YANG data models) make a clear distinction between configuration data and state data. SJA1105 port counters are an example of state data exported by the yang-sja1105 netopeer module. These can be very useful for debugging or investigating the traffic remotely.

```
# Get all configuration + state data
# get-config, by contrast, shows just configuration data
netconf> get
# Get just the port counters
netconf> get --filter
<sja1105>
    <ports/>
</sja1105>
```

7.11.7 Ending the NETCONF session

Use the following command to end the NETCONF session:

```
netconf> disconnect
```

Chapter 8
4G-LTE Modem

8.1 Introduction

4G-LTE USB modem is supported.

8.2 Hardware preparation

A HuaWei E3372 USB Modem (as example) is used for the 4G-LTE network verification.

Insert this USB modem into USB slot of LS1012ARDB board (LS1012ARDB as example).

8.3 Software preparation

In order to support 4G-LTE modem, some options are needed.

1. In OpenIL environment, use command “make menuconfig” to enable the below options:

```
$make menuconfig
System configuration  --->
        <*> /dev management (Dynamic using devtmpfs + eudev)

Target packages  --->
 Hardware handling  --->
            <*> usb_modeswitch
<*> usb_modeswitch_data
```

2. In Linux kernel environment, make sure the below options are enabled:

```
$make linux-menuconfig
Device Drivers  --->
     [*] Network device support  --->
<*>   USB Network Adapters --->
<*>   CDC Ethernet support
<*>   CDC EEM support
<*>   CDC NCM support
```

   Finally, update the images, refer to section 3.4.2 (LS1012ARDB as example).

8.4 Testing 4G USB modem link to the internet

Perform the following instructions to set up the 4G Modem .

After booting up the Linux kernel, an Ethernet interface will be identified, for example “eth2”.

Testing 4G USB modem link to the internet

1. Set eth2 connected to the network.

```
$ udhcpc -BFs -i eth2
```

2. Test the 4G modem link to the internet.

```
$ ping www.nxp.com
PING www.nxp.com (210.192.117.231): 56 data bytes
64 bytes from 210.192.117.231: seq=0 ttl=52 time=60.223 ms
64 bytes from 210.192.117.231: seq=1 ttl=52 time=95.076 ms
64 bytes from 210.192.117.231: seq=2 ttl=52 time=89.827 ms
64 bytes from 210.192.117.231: seq=3 ttl=52 time=84.694 ms
64 bytes from 210.192.117.231: seq=4 ttl=52 time=68.566 ms
```

Chapter 9
OTA implementation

NXP's LS1021-IoT, LS1012ARDB, LS1043ARDB, and LS1046ARDB platforms support OTA (Over-the-air) requirements. This section provides an introduction to OTA use cases, scripts, configuration settings for implementation and server preparation, and a test case. It also lists the OTA features supported by each hardware platform.

9.1 Introduction

OTA refers to a method of updating U-Boot, kernel, file system, and even the full firmware to devices through the network. If the updated firmware does not work, the device can rollback the firmware to the latest version automatically.

NOTE

While updating U-Boot, there is no hardware method to rollback the device automatically, hence the device might not be rolled back, once the U-Boot is not working.

- **version.json**: This is a JSON file which saves the board name and version of each firmware. Below is an example of version.json.

```
{
"updatePart":"kernel", /* Name of firmware image which has been updated. */
"updateVersion":"1.0", /* Version of firmware image which has been updated. */
"all":"1.0", /* version of the full firmware image which has been used now */
"u-boot":"1.0", /* version of the u-boot image which has been used now */
"kernel":"1.0", /* version of the kernel image which has been used now */
"filesystem":"1.0", /* version of the filesystem image which has been used now */
"boardname":"ls1021aiot" /* used to get the corresponding firmware from server*/
"URL":"https://www.nxp.com/lgfiles/iiot" /* used to get the corresponding firmware from
server*/
}
```

- **update.json**: This file is stored in server, it saves the name and version of firmware image which will be updated. Below is a sample update.json file:

```
{
"updateStatus":"yes", /* set yes or no to tell devices is it need to update. */
"updatePart":"kernel", /* name of update firmware. */
"updateVersion":"1.0", /* version of update firmware */
}
```

- **ota-update**: This script can get a JSON file named update.json from server, then parse the file and get the new firmware version to confirm whether to download it from server or not. It finally writes the firmware into the SD card instead of the old one. After that, save the "updatePart" and "updateVersion" into version.json, and mark the update status on 4080 block of SD card to let U-Boot know it.
- **ota-versioncheck**: This script checks if the firmware has been updated, then updates the version of the update part in version.json, and cleans the flag of update status on 4080 block of SD card. This script runs automatically each time the system restarts.
- **ota-rollback**: This script runs on the ramdisk filesystem after the filesystem update fails. It gets the old firmware version from the version.json file and then updates it from the server.

9.2 Platform support for OTA demo

The OTA demo is supported by four NXP hardware platforms. Following is the list of features supported by each platform:

1. **LS1021A-IoT**
   - Full SD card firmware update
   - U-Boot image update kernel image update
   - File system image update
   - Full SD card firmware update
2. **LS1012ARDB**
   - Full SD card firmware update
   - RCW and U-Boot image update on QSPI flash
   - Kernel image update and rollback
   - File system image update and rollback
3. **LS1043ARDB**
   - Full SD card firmware update
   - U-Boot image update
   - Kernel image update and rollback
   - File system image update and rollback
4. **LS1046ARDB**
   - Full SD card firmware update
   - U-Boot image update
   - Kernel image update and rollback
   - File system image update and rollback

9.3 Server requirements

This demo provides a sample server to update images for the v1.0 release. In case you want to use another server, you need to change the URL to your own server path at `"target/linux/layerscape/image/backup/version.json"` such as the following:

```
"URL":"https://www.nxp.com/lgfiles/iiot/"
```

The server must include a JSON file named `update.json` that can send information to device boards. Below is a sample `update.json` file.

```
{
    /* set yes or no to tell devices is it need to update. */
        "updateStatus":"yes",

    /* which part to update, you can write "all", "u-boot", "kernel",  "filesystem"  */
        "updatePart":"filesystem",

    /* version of update firmware */
```

```
        "updateVersion":"1.0",
}
```

Images for OTA are stored in the path:

<updateVersion>/<boardname>/

where the <boardname> can be one of these: `ls1021aiot`, `ls1012ardb-64b`, `ls1012ardb-32b`, `ls1043ardb-64b`, `ls1043ardb-32b`, `ls1046ardb-64b`, or `ls1046ardb-32b`.

Images must be named as following:

- `u-boot.bin`: U-Boot image for update. In `ls1012ardb` folder, this image includes RCW and U-Boot.
- `uImage`: kernel image for update
- `rootfs.ext4`: filesystem image for update
- `firmware_sdcard.bin`: a full firmware of SD card image.

9.4 OTA test case

1. Plug network cable into Eth1 on the board. This enables the network after the system is running.
2. Update U-Boot using the following steps:
   - Update the .json on server as shown in the following example:

     ```
     {
                          "updateStatus":"yes",
                         "updatePart":"u-boot",
                          "updateVersion":"1.0",
     }
     ```

   - Upload the u-boot image on server path: 1.0/<boardname>/u-boot.bin
   - Run ota-update command on device board.
3. Updating the file system:
   - Set the "updatePart" to "filesystem" in update.json.
   - Upload the filesystem image on server path: 1.0/<boardname>/rootfs.ext4
   - Run ota-update command on the device board.
4. Updating full firmware
   - Set the "updatePart" to "all" in update.json.
   - Upload the full firmware image on server path: 1.0/<boardname>/firmware_sdcard.bin
   - Run ota-update command on device board.
5. Rollback test:
   - The Kernel and file system can use a wrong image to upload on the server and test update on device.

Chapter 10
EtherCAT

OpenIL supports the use of EtherCAT ((Ethernet for Control Automation Technology) and integrates the IGH EtherCAT master stack. EtherCAT support is verified on NXP's LS1021-IoT, LS1043ARDB, and LS1046ARDB platforms.

10.1 Introduction

EtherCAT is an Ethernet-based fieldbus system, invented by BECKHOFF Automation. The protocol is standardized in IEC 61158 and is suitable for both hard and soft real-time computing requirements in automation technology. The goal during development of EtherCAT was to apply Ethernet for automation applications requiring short data update times (also called cycle times; ≤ 100 µs) with low communication jitter (for precise synchronization purposes; ≤ 1 µs) and reduced hardware costs.

- EtherCAT is Fast: 1000 dig. I/O: 30 µs, 100 slaves: 100 µs.
- EtherCAT is Ethernet: Standard Ethernet at I/O level.
- EtherCAT is Flexible: Star, line, drop, with or without switch.
- EtherCAT is Inexpensive: ethernet is mainstream technology, therefore inexpensive.
- EtherCAT is Easy: everybody knows Ethernet, it is simple to use.

At present, the EtherCAT master supports the common open source code for SOEM of RT - LAB development (Simple Open Source EtherCAT Master) and EtherLab, the IGH EtherCAT master. To use SOEM is simpler than to use the IGH EtherCAT Master, but IGH for the realization of the EtherCAT is more complete. For example, IGH supports more NIC. For more information, see https://rt-labs.com/ethercat/ and http://www.etherlab.org. The integration in OpenIL is IGH EtherCAT master.

10.2 IGH EtherCAT architecture

The components of the master environment are described below:

- **Master module:** This is the kernel module containing one or more EtherCAT master instances, the 'Device Interface' and the 'Application Interface'.
- **Device modules:** These are EtherCAT-capable Ethernet device driver modules that offer their devices to the EtherCAT master via the device interface. These modified network drivers can handle network devices used for EtherCAT operation and 'normal' Ethernet devices in parallel. A master can accept a certain device and then, is able to send and receive EtherCAT frames. Ethernet devices declined by the master module are connected to the kernel's network stack, as usual.
- **Application:** A program that uses the EtherCAT master (usually for cyclic exchange of process data with EtherCAT slaves). These programs are not part of the EtherCAT master code, but need to be generated or written by the user. An application can request a master through the application interface. If this succeeds, it has the control over the master: It can provide a bus configuration and exchange process data. Applications can be kernel modules (that use the kernel application interface directly) or user space programs, that use the application interface via the EtherCAT library, or the RTDM library.

  The following figure shows that IGH EtherCAT master architecture.

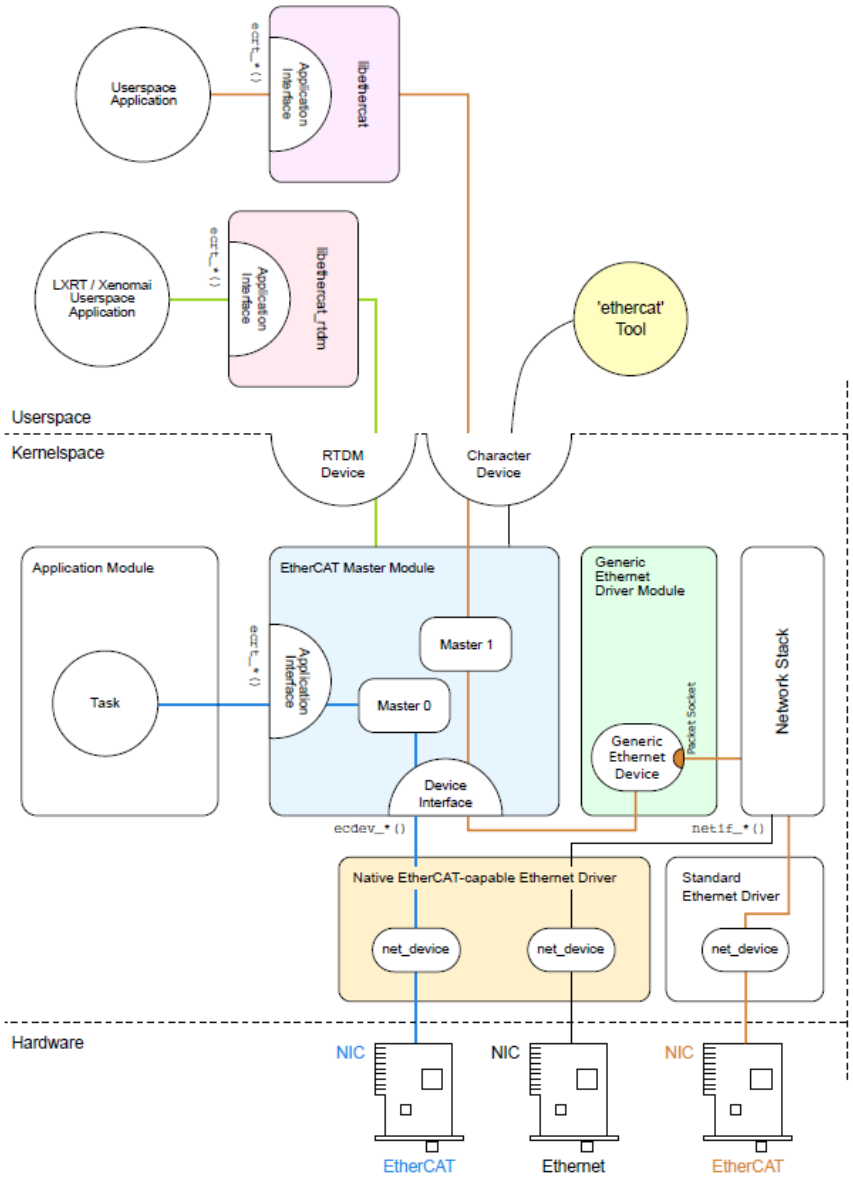


Figure 57. IGH EtherCAT master architecture

10.3 EtherCAT protocol

Following are the characteristics of the EtherCAT protocol:

- The EtherCAT protocol is optimized for process data and is transported directly within the standard IEEE 802.3 Ethernet frame using Ethertype 0x88a4.
- The data sequence is independent of the physical order of the nodes in the network; addressing can be in any order.
- Broadcast, multicast, and communication between slaves is possible, but must be initiated by the master device.
- If IP routing is required, the EtherCAT protocol can be inserted into UDP/IP datagrams. This also enables any control with Ethernet protocol stack to address EtherCAT systems.
- It does not support shortened frames.

The following figure shows the EtherCAT frame structure.

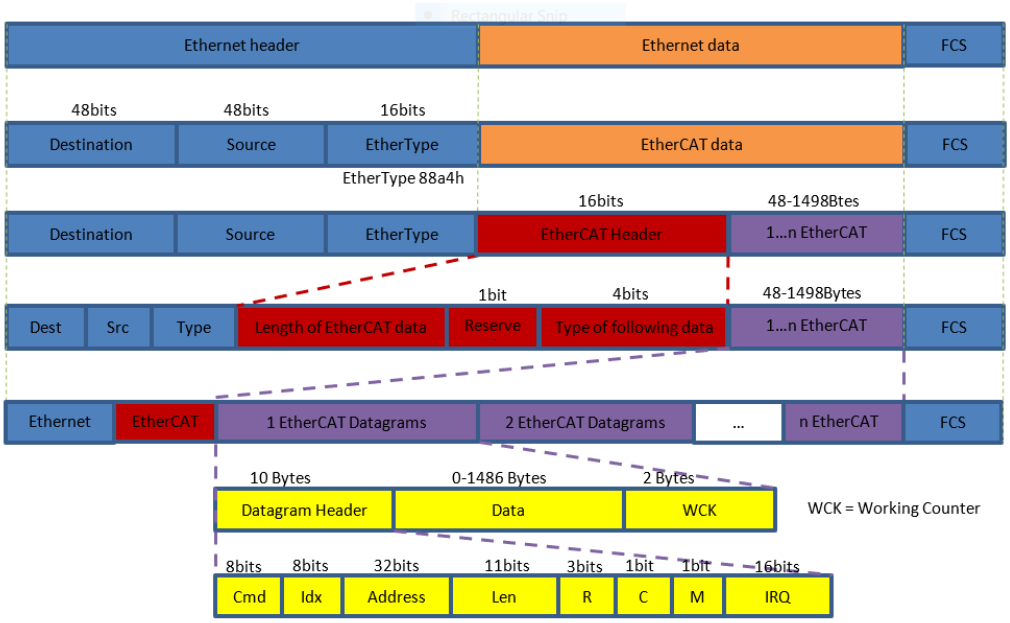


Figure 58. EtherCAT frame structure

10.4 EtherCAT system integration and example

This section describes how to integrate EtherCAT with the OpenIL system and provides an example of running the BECKHOFF application.

10.4.1 Building kernel images for EtherCAT

For **LS1021A-IoT**, EtherCAT supports the following configuration files:

- nxp_ls1021aiot_baremetal_defconfig
- nxp_ls1021aiot_baremetal_ubuntu_defconfig
- nxp_ls1021aiot_defconfig
- nxp_ls1021aiot_optee_defconfig
- nxp_ls1021aiot_optee_ubuntu_defconfig
- nxp_ls1021aiot_ubuntu_defconfig.

For **LS1043ARDB**, EtherCAT supports the following configurations:

- nxp_ls1043ardb-64b_defconfig
- nxp_ls1043ardb-64b_ubuntu_defconfig
- nxp_ls1043ardb_baremetal-64b_defconfig.

For **LS1046ARDB**, EtherCAT supports the following configurations:

- nxp_ls1046ardb-64b_defconfig
- nxp_ls1046ardb-64b_qspi_defconfig
- nxp_ls1046ardb-64b_qspi-sb_defconfig

- nxp_ls1046ardb-64b_ubuntu_defconfig
- nxp_ls1046ardb_baremetal-64b_defconfig.

Use the command below to build image supporting EtherCAT (example: nxp_ls1046ardb-64b_defconfig):

```
$ make nxp_ls1046ardb-64b_defconfig
$ make
```

Then, flash the image to SD card and reboot the board with this card and SD boot.

10.4.2 Command-line tool

Each master instance gets a character device as a userspace interface. The devices are named `/dev/EtherCATx`, where `x` is the index of the master.

Device node creation The character device nodes are automatically created, if the startup script is executed. The following example illustrates the command-line tools:

Table 20. Command line tools for EtherCAT

Command	Description	Arguments	Output
ethercat config [OPTIONS]	Shows slave configurations.	Options: • --alias -a <alias > Configuration alias (see above) • --position -p <pos > Relative position (see above). • -- verbose -v Show detailed configurations.	Without the -- verbose option, slave configurations are output one -per - line. For example, the output for1001:0 0 x0000003b / 0 x02010000 3 would be displayed as follows: • **1001:0** -> Alias address and relative position (both decimal). • **0 x0000003b /0 x02010000** -> Expected vendor ID and product code (both hexadecimal). • **3** -> Absolute decimal ring position of the attached slave, or '-' if none attached. • **OP** -> Application – layer state of the attached slave, or '-', if no slave is attached.

Table continues on the next page...

Table 20. Command line tools for EtherCAT (continued)

ethercat master [OPTIONS]	Shows master and Ethernet device information.	Options: -- master -m <indices > Master indices. A comma - separated list with ranges is supported. Example: 1 ,4 ,5 ,7 -9. Default: - (all).	Master0 Phase: Idle Active: no Slaves: 8 Ethernet devices: Main: 00:00:08:44: ab :66 (attached) Link: UP Tx frames: 18846 Tx bytes: 1169192 Rx frames: 18845 Rx bytes: 1169132 Tx errors: 0 Tx frame rate [1/s]: 125 395 241 Tx rate [KByte/s]: 7.3 24.0 14.6 Rx frame rate [1/s]: 125 395 241 Rx rate [KByte/s]: 7.3 24.0 14.6 Common: Tx frames: 18846 Tx bytes: 1169192 Rx frames: 18845 Rx bytes: 1169132 Lost frames: 0 Tx frame rate [1/s]: 125 395 241 Tx rate [KByte/s]: 7.3 24.0 14.6 Rx frame rate [1/s]: 125 583 241 Rx rate [KByte/s]: 7.3 210.4 14.6 Loss rate [1/s]: 0 -0 0 Frame loss [%]: 0.0 -0.0 0.0 Distributed clocks: Reference clock: Slave 0 Application time: 0

Table continues on the next page...

Table 20. Command line tools for EtherCAT (continued)

ethercat states [OPTIONS] <STATE >	Requests application - layer states	STATE can be 'INIT ', 'PREOP ', 'BOOT ', 'SAFEOP ', or 'OP '. Options: • --alias -a <alias > • -- position -p <pos > Slave selection. See the help of the 'slaves' command.	None

NOTE

- Numerical values can be specified either with decimal (no prefix), octal (prefix '0') or hexadecimal (prefix '0x ') base.
- More command-line information can be obtained by using the command **ethercat --help**.

10.4.3 System integration

An `init` script and a `sysconfig` file are provided to integrate the EtherCAT master as a service into a running system. These are described below.

- **Init Script**

The EtherCAT master `init` script conforms to the requirements of the 'Linux Standard Base' (LSB). The script is installed to `etc/init.d/EtherCAT`, before the master can be inserted as a service. Please note, that the `init` script depends on the `sysconfig` file described below.

LSB defines a special comment block to provide service dependencies (that is, which services should be started before others) inside the init script code. System tools can extract this information to insert the EtherCAT init script at the correct place in the startup sequence:

```
    # Required - Start: $local_fs $syslog $network
    # Should - Start: $time ntp
    # Required - Stop: $local_fs $syslog $network
    # Should - Stop: $time ntp
    # Default - Start: 3 5
    # Default - Stop: 0 1 2 6
    # Short - Description: EtherCAT master
    # Description: EtherCAT master 1.5.2
    ### END INIT INFO
```

- **Sysconfig file**

For persistent configuration, the `init` script uses a `sysconfig` file installed to `etc/sysconfig/EtherCAT`, that is mandatory for the `init` script. The sysconfig file contains all configuration variables needed to operate one or more masters. The documentation is inside the file and included below:

```
#------------------------------------------------------------------------------
## Main Ethernet devices.
#
# The MASTER <X> _DEVICE variable specifies the Ethernet device for a master
# with index 'X '.
#
```

```
# Specify the MAC address (hexadecimal with colons) of the Ethernet device to
# use. Example: "00:00:08:44: ab :66"
#
# The broadcast address "ff:ff:ff:ff:ff:ff" has a special meaning : It tells
# the master to accept the first device offered by any Ethernet driver.
#
# The MASTER <X> _DEVICE variables also determine, how many masters will be
# created: A non - empty variable MASTER0_DEVICE will create one master, adding a
# non - empty variable MASTER1_DEVICE will create a second master, and so on.
#
MASTER0_DEVICE =""
# MASTER1_DEVICE =""
#
# Backup Ethernet devices
#
# The MASTER <X> _BACKUP variables specify the devices used for redundancy. They
# behaves nearly the same as the MASTER <X> _DEVICE variable, except that it
# does not interpret the ff:ff:ff:ff:ff:ff address .
#
# MASTER0_BACKUP =""
#
# Ethernet driver modules to use for EtherCAT operation.
#
# Specify a non - empty list of Ethernet drivers, that shall be used for
# EtherCAT operation.
#
# Except for the generic Ethernet driver module, the init script will try to
# unload the usual Ethernet driver modules in the list and replace them with
# the EtherCAT - capable ones. If a certain (EtherCAT - capable) driver is not
# found, a warning will appear.
#
# Possible values: 8139 too, e100, e1000, e1000e, r8169, generic, ccat, igb.
# Separate multiple drivers with spaces.
#
# Note: The e100, e1000, e1000e, r8169, ccat and igb drivers are not built by
# default. Enable them with the --enable -<driver > configure switches.
#
# Attention: When using the generic driver, the corresponding Ethernet device
# has to be activated (with OS methods, for example 'ip link set ethX up '),
# before the master is started, otherwise all frames will time out.
#
DEVICE_MODULES =""
#
# Flags for loading kernel modules.
#
# This can usually be left empty. Adjust this variable, if you have problems
# with module loading.
#
# MODPROBE_FLAGS ="-b"
#-------------------------------------------------------------------------------
```

Starting the Master as a service: After the `init` script and the `sysconfig` file are placed into the right location, the EtherCAT master can be inserted as a service.The `init` script can also be used for manually starting and stopping the EtherCAT master. It should be executed with one of the parameters: start, stop, restart or status. For example:

```
$/etc/init.d/EtherCAT restart
    Shutting down EtherCAT master done
    Starting EtherCAT master done
```

10.4.4 Running a sample application

This section describes how to run a sample application.

List of materials

Following is the list of materials needed for running the Igh EtherCAT application:

- OpenIL board (LS1021-IoT, LS1043ARDB, and LS1046ARDB)
- BECKHOFF EK1100 and EL2008
- 42-stepping motor and stepper motor driver

The figures below show the required materials:

- The figure below shows the board and BECKHOFF connected by a wire.

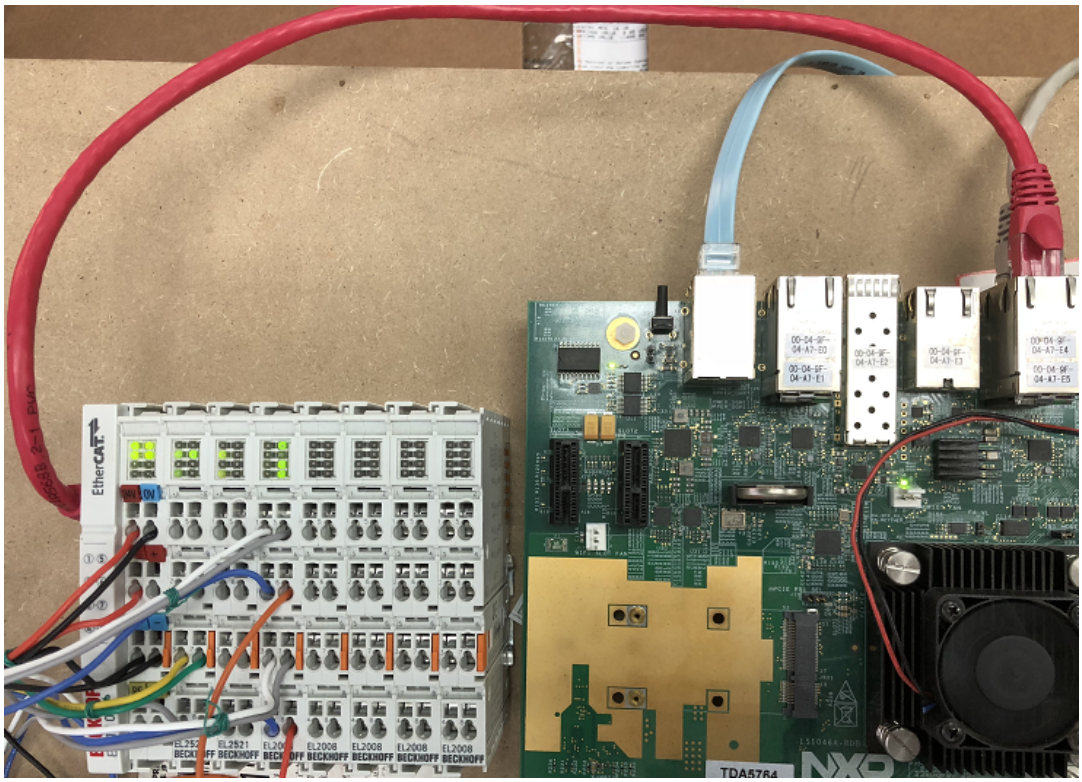


Figure 59. Board connects with BECKHOFF

- The figure below shows the BECKHOFF's EK1100 and EL2008.

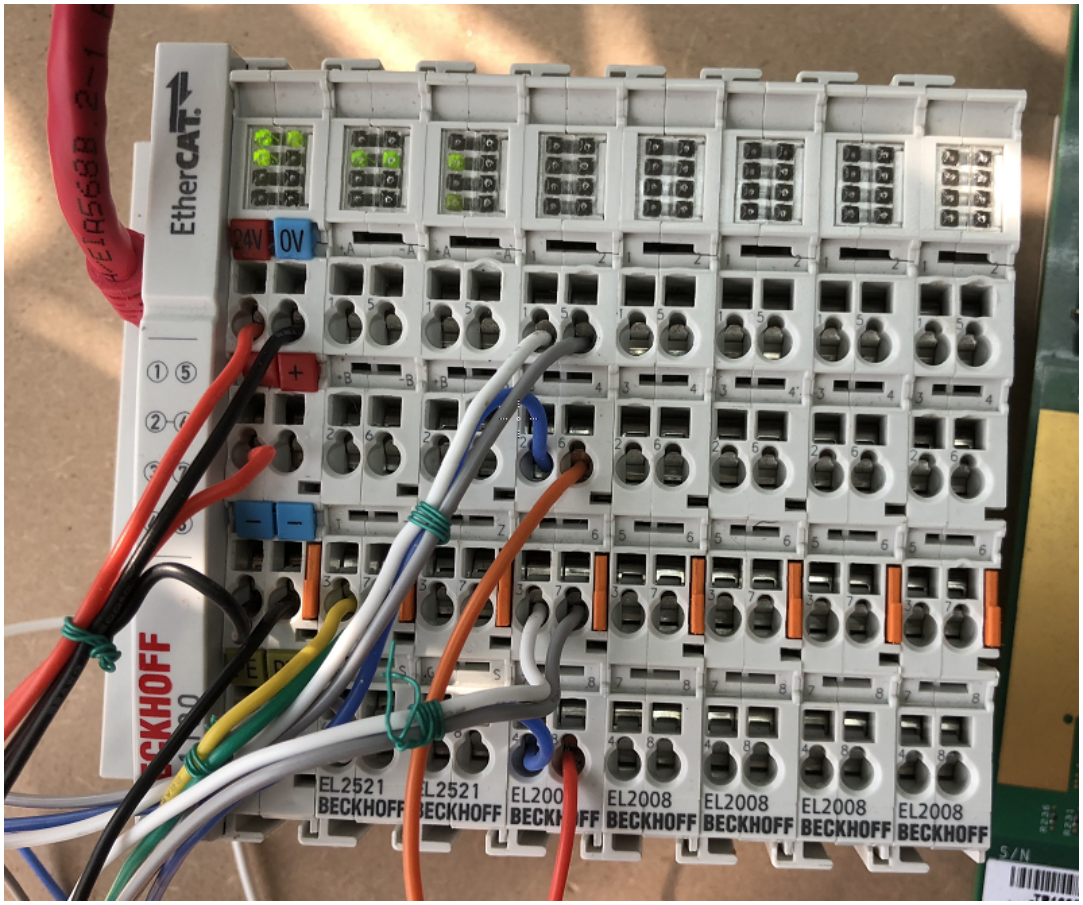


Figure 60. BECKHOFF EL2008

- The figure below shows a stepper motor driver.

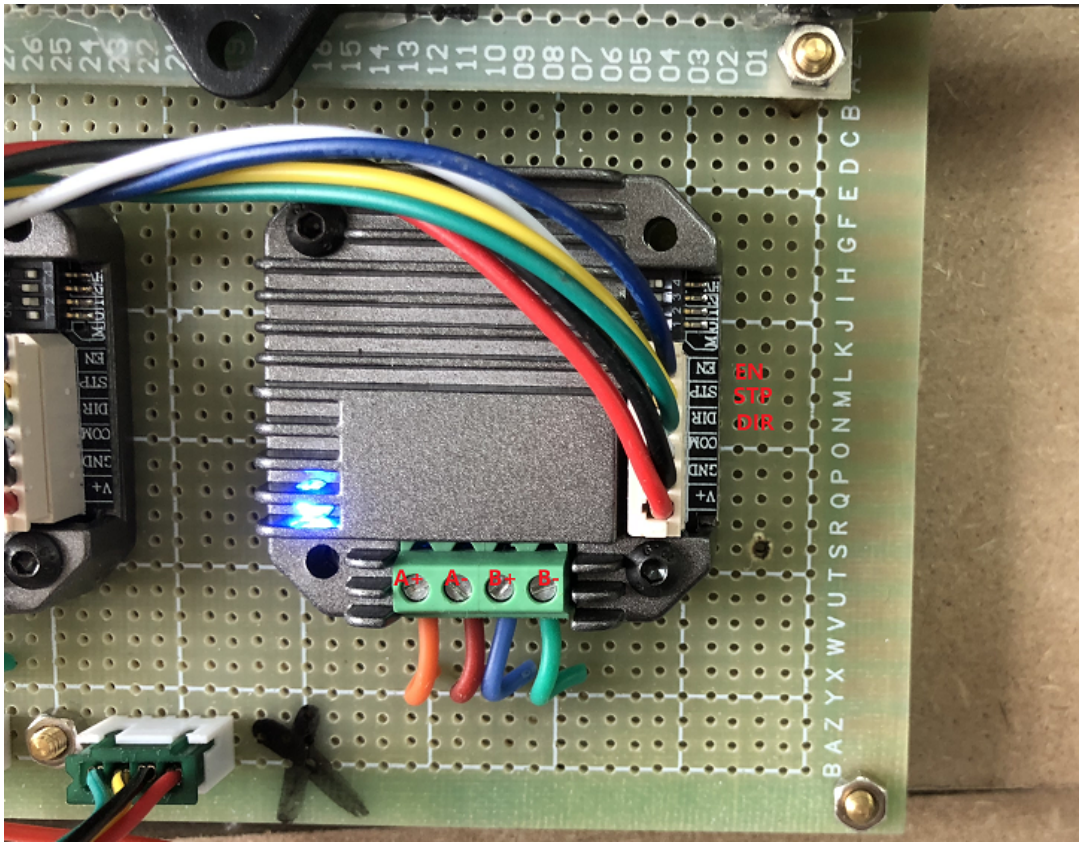


Figure 61. Stepper motor driver

The stepper motor needs to be connected to the EL2008 with a driver.

EL2008 needs connections to the EN, STP, and DIR pins of the stepping motor drive.

- The figure below shows a 42-stepper motor. Note the manner in which the stepper motor is connected to the driver:
  - A is connected to A.
  - B is connected to B.

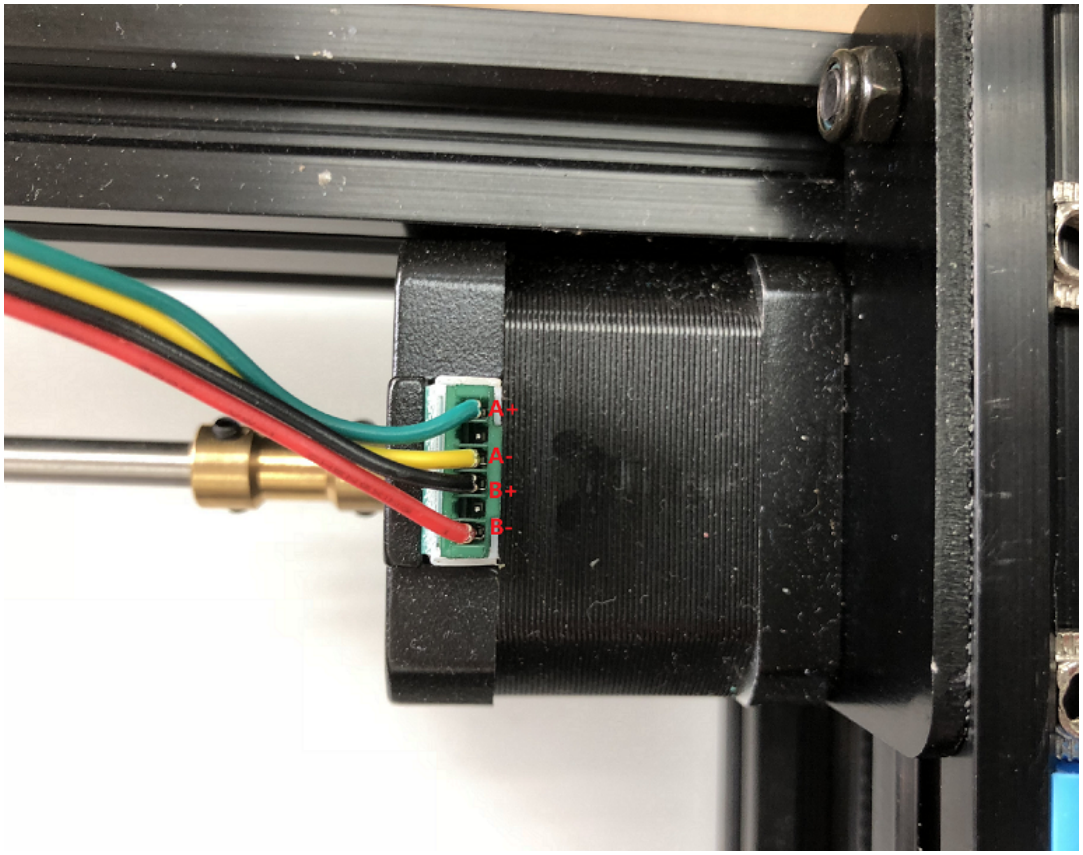


Figure 62. Stepper motor

For more information about EL2008, see https://www.beckhoff.com/english.asp?ethercat/el2008.htm.

Follow the steps below to run a sample application:

1. Update the `sysconfig file etc/sysconfig/EtherCAT` for the persistent configuration.Variables `MASTER0_DEVICE` and `DEVICE_MODULES` need to be changed to the specified MAC and driver type. The MAC address is the one that is connected to BECKHOFF.

   For example, the MAC used is `00:00:08:44: ab :66` and the drivers used are `generic`:

   ```
   MASTER0_DEVICE ="00:00:08:44: ab :66"
   DEVICE_MODULES ="generic"
   ```

2. Execute the initialization script and specify the parameter start.

   ```
   $ /etc/init.d/ethercat restart
   ```

3. Run the example application.

   ```
   $ ec_user_example
   ```

   - If the `init` script fails to start EtherCAT master, the command `insmod` or `modprobe` can be used to load the module directly: `ec_master.ko` and `ec_generic.ko` are found in the path `/lib/modules/4.9.35-ipipe/extra/`

     ```
     $ insmod ec_master.ko main_devices= MAC address
     $ insmod ec_generic.ko
     ```

- Run the example application.

```
$ ec_user_example
```

ATTENTION

If the console prompts `Failed to open /dev/EtherCAT0`, the module fails to load, please check it.

Chapter 11
FlexCAN

The following sections provide an introduction to the FlexCAN standard, details of the CAN bus, the Canopen communication system, details of how to integrate FlexCAN with OpenIL, and running a FlexCAN application.

11.1 Introduction

The LS1021A has the FlexCAN module. The FlexCAN module is a communication controller implementing the CAN protocol according to the CAN 2.0 B protocol specification. The main sub-blocks implemented in the FlexCAN module include an associated memory for storing message buffers, Receive (Rx) Global Mask registers, Receive Individual Mask registers, Receive FIFO filters, and Receive FIFO ID filters. A general block diagram is shown in the following figure. The functions of these submodules are described in subsequent sections.

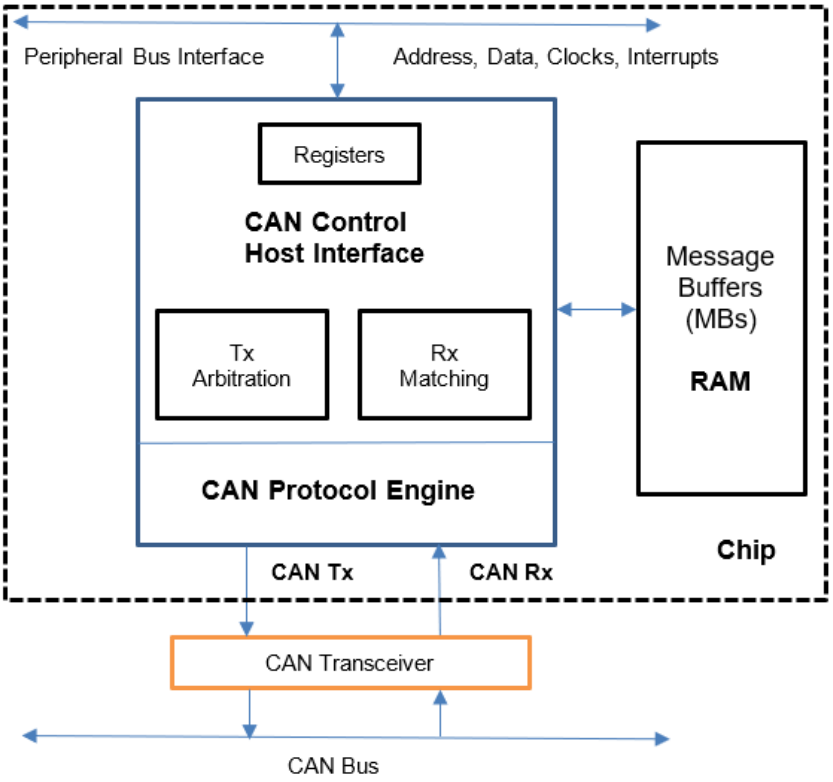


Figure 63. FlexCAN block diagram

11.1.1 CAN bus

CAN (Controller Area Network) is a serial bus system. A CAN bus is a robust vehicle bus standard designed to allow microcontrollers and devices to communicate with each other in applications without a host computer. Bosch published several versions of the CAN specification and the latest is CAN 2.0 published in 1991. This specification has two parts; part A is for the standard format with an 11-bit identifier, and part B is for the extended format with a 29-bit identifier. A CAN device that uses 11-bit identifiers is commonly called CAN 2.0A and a CAN device that uses 29-bit identifiers is commonly called CAN 2.0B.

CAN is a multi-master serial bus standard for connecting Electronic Control Units [ECUs] also known as nodes. Two or more nodes are required on the CAN network to communicate. The complexity of the node can range from a simple I/O device up to an embedded computer with a CAN interface and sophisticated software. The node may also be a gateway allowing a standard computer to communicate over a USB or Ethernet port to the devices on a CAN network. All nodes are connected to each other through a two wire bus. The wires are a twisted pair with a 120 Ω (nominal) characteristic impedance.

High speed CAN signaling drives the CAN high wire towards 5 V and the CAN low wire towards 0 V when transmitting a dominant (0), and does not drive either wire when transmitting a recessive (1). The dominant differential voltage is a nominal 2 V. The termination resistor passively returns the two wires to a nominal differential voltage of 0 V. The dominant common mode voltage must be within 1.5 to 3.5 V of common and the recessive common mode voltage must be within +/-12 of common.

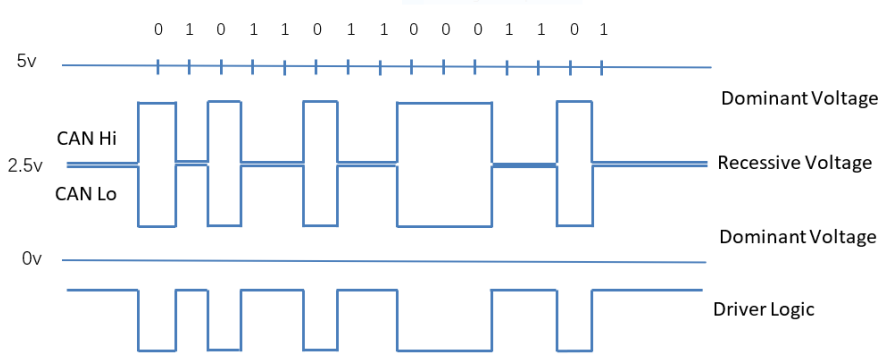


Figure 64. High speed CAN signaling

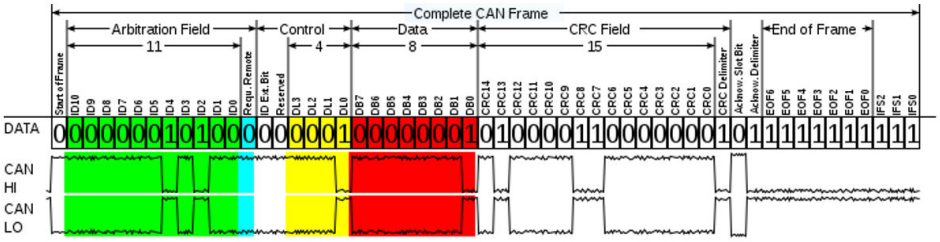


Figure 65. Base frame format

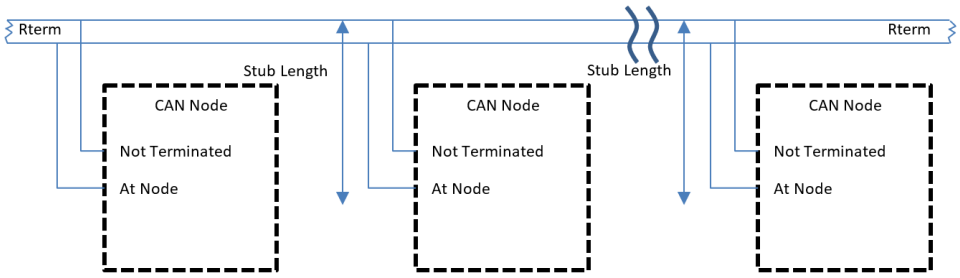


Figure 66. High speed CAN network

11.1.2 CANopen

CANopen is a CAN-based communication system. It comprises higher-layer protocols and profile specifications. CANopen has been developed as a standardized embedded network with highly flexible configuration capabilities. Today it is used in various application fields, such as medical equipment, off-road vehicles, maritime electronics, railway applications, or building automation.

CANopen provides several communication objects, which enable device designers to implement desired network behavior into a device. With these communication objects, device designers can offer devices that can communicate process data, indicate device-internal error conditions or influence and control the network behavior. As CANopen defines the internal device structure, the system designer knows exactly how to access a CANopen device and how to adjust the intended device behavior.

- **CANopen lower layers**

  CANopen is based on a data link layer according to ISO 11898-1. The CANopen bit timing is specified in CiA 301 and allows the adjustment of data rates from 10 kbit/s to 1000 kbit/s. Although all specified CAN-ID addressing schemata are based on the 11-bit CAN-ID, CANopen supports the 29-bit CAN-ID as well. Nevertheless, CANopen does not exclude other physical layer options.

- **Internal device architecture**

  A CANopen device consists of three logical parts. The CANopen protocol stack handles the communication via the CAN network. The application software provides the internal control functionality. The CANopen object dictionary interfaces the protocol as well as the application software. It contains indices for all used data types and stores all communication and application parameters. The CANopen object dictionary is most important for CANopen device configuration and diagnostics.

- **CANopen protocols**
  - SDO protocol
  - PDO protocol
  - NMT protocol
  - Special function protocols
  - Error control protocols

The following figure shows the CANopen architecture.

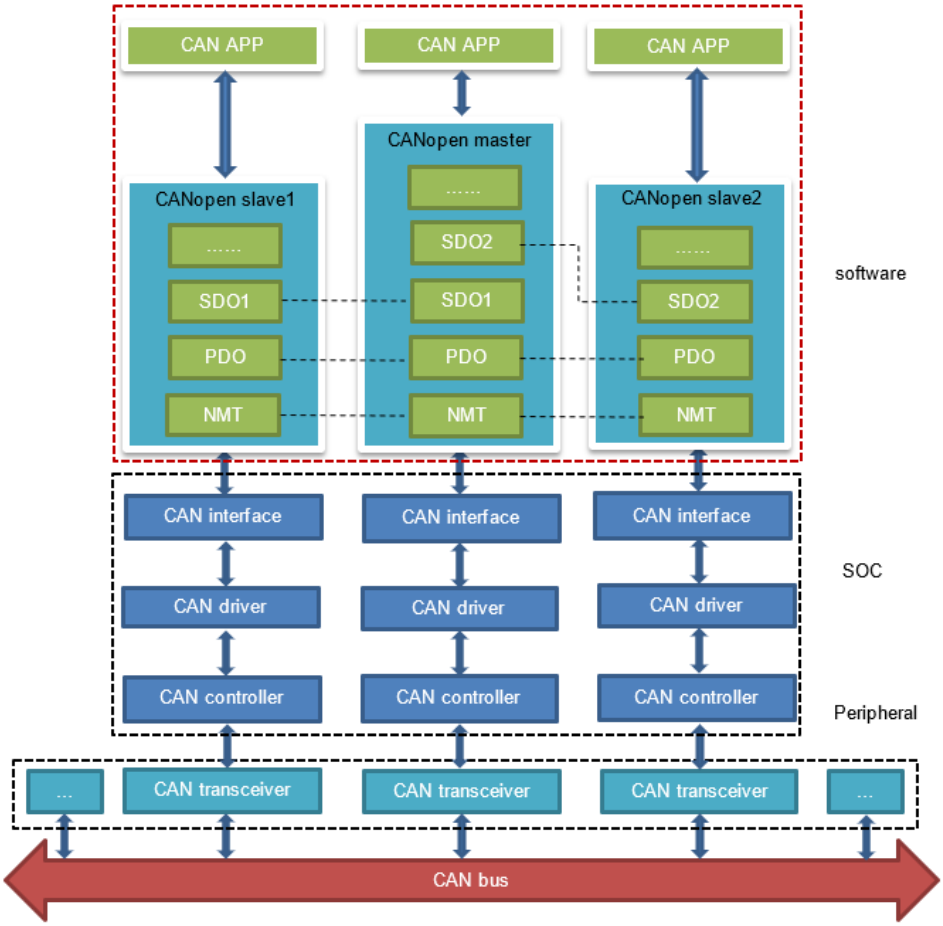


Figure 67. CANopen architecture

11.2 FlexCAN integration in OpenIL

Two CAN controllers (CAN3 and CAN4) are used to communicate with each other. CAN4 is assigned to core0, which runs Linux and CANOpen as master node, whereas CAN3 is assigned to core1, which runs the baremetal and CANOpen as slave node.

11.2.1 Resource allocation

This section describes steps for assigning CAN4 to Linux and CAN3 to baremetal core, and how to change or configure it. These examples assume that CAN1 and CAN2 are not enabled, and the pins of CAN1 and CAN2 are used by other IPs.

1. **Assigning CAN4 to Linux**

   In Linux, the port is allocated through the DTS file. DTS file path is `industry-linux/arch/arm/boot/dts/ls1021a-iot.dts`. Content related to CAN ports is as follows:

```
/* CAN3 port */
    &can2
        {
           status = " disabled ";
        };
```

```
    /* CAN4 port */
        &can3
            {
                status = "okay";
            };
```

2. **Assigning CAN3 to Baremetal**

   In baremetal, the port is allocated through the `flexcan.c` file. The flexcan.c path is `industry-uboot/drivers/flexcan/flexcan.c`. In this file, you need to define the following variables:

   a. `struct can_bittiming_t flexcan3_bittiming = CAN_BITTIM_INIT(CAN_500K);`

   NOTE

   Set bit timing and baud rate (500K) of the CAN port.

   b. struct can_ctrlmode_t flexcan3_ctrlmode

   ```
   struct can_ctrlmode_t flexcan3_ctrlmode =
   {
       .loopmode = 0, /* Indicates whether the loop mode is enabled */
       .listenonly = 0, /* Indicates whether the only-listen mode is enabled */
       .samples = 0,
       .err_report = 1,
   };
   ```

   c. struct can_init_t flexcan3

   ```
   struct can_init_t flexcan3 =
   {
       .canx = CAN3, /* Specify CAN port */
       .bt = &flexcan3_bittiming,
       .ctrlmode = &flexcan3_ctrlmode,
       .reg_ctrl_default = 0,
       .reg_esr = 0
   };
   ```

   d. Optional parameters

   - **CAN port**

     ```
     #define CAN3 ((struct can_module *)CAN3_BASE)
     #define CAN4 ((struct can_module *)CAN4_BASE)
     ```

   - **Baud Rate**

     ```
     #define CAN_1000K 10
     #define CAN_500K 20
     #define CAN_250K 40
     #define CAN_200K 50
     #define CAN_125K 80
     #define CAN_100K 100
     #define CAN_50K 200
     #define CAN_20K 500
     #define CAN_10K 1000
     #define CAN_5K 2000
     ```

11.2.2 Introducing the function of CAN example code

CAN example code supports the CANopen protocol. It mainly implements three parts of functions: network manage function (NMT protocol), service data transmission function (SDO protocol), and process data transmission function (PDO protocol). NMT protocol can manage and monitor slave nodes, include heart beat message. SDO protocol can transmit single or block data. The PDO protocol can transmit process data that requires real time.

CAN example calls the CANopen interfaces, described in the table below:

Table 21. CAN Net APIs and their description

API name (type)	Description
UNS8 canReceive_driver (CAN_HANDLE fd0, Message * m)	Socketcan receive CAN messages • fd0 – socketcan handle • m – receive buffer
UNS8 canSend_driver (CAN_HANDLE fd0, Message const * m)	Socketcan send CAN messages • fd0 – socketcan handle • m – CAN message to be sent
void setNodeId(CO_Data* d, UNS8 nodeId)	Set this node id value. • d – object dictionary • nodeId – id value (up to 127)
UNS8 setState(CO_Data* d, e_nodeState newState)	Set node state • d – object dictionary • newState – The state that needs to be set Returns 0 if ok, > 0 on error
void canDispatch(CO_Data* d, Message *m)	CANopen handles data frames that CAN receive. • d – object dictionary • m – Received CAN message
void timerForCan(void)	CANopen virtual clock counter.
UNS8 sendPDOrequest (CO_Data * d, UNS16 RPDOIndex)	Master node requests slave node to feedback specified data. • d – object dictionary • RPDOIndex – index value of specified data

Table continues on the next page...

Table 21. CAN Net APIs and their description (continued)

API name (type)	Description
UNS8 readNetworkDictCallback (CO_Data* d, UNS8 nodeId, UNS16 index, UNS8 subIndex, UNS8 dataType, SDOCallback_t Callback, UNS8 useBlockMode)	The master node gets the specified data from the slave node. • d – object dictionary • nodeId – the id value of slave node • index – the index value of the specified data • subIndex – the subindex value of the specified data • dataType – the data type of the specified data • Callback – callback function • useBlockMode – specifies whether it is a block transmission
UNS8 writeNetworkDictCallBack (CO_Data* d, UNS8 nodeId, UNS16 index, UNS8 subIndex, UNS32 count, UNS8 dataType, void *data, SDOCallback_t Callback, UNS8 useBlockMode)	The master node sets the specified data to the slave node. • d – object dictionary • nodeId – the id value of slave node • index – the index value of the specified data • subIndex – the subindex value of the specified data • count – the length of the specified data • dataType – the data type of the specified data • Callback – callback function • useBlockMode – specifies whether it is a block transmission

11.3 Running a CAN application

The following sections describe the hardware and software preparation steps for running a CAN application.

11.3.1 Hardware preparation

The list of hardware required for implementing the FlexCAN demo is as follows:

- LS1021A-IoT boards
- Two CAN hardware interfaces (for example, CAN3 and CAN4 for LS1021A-IoT)
- Two CAN transceivers (for example: TJA1050)

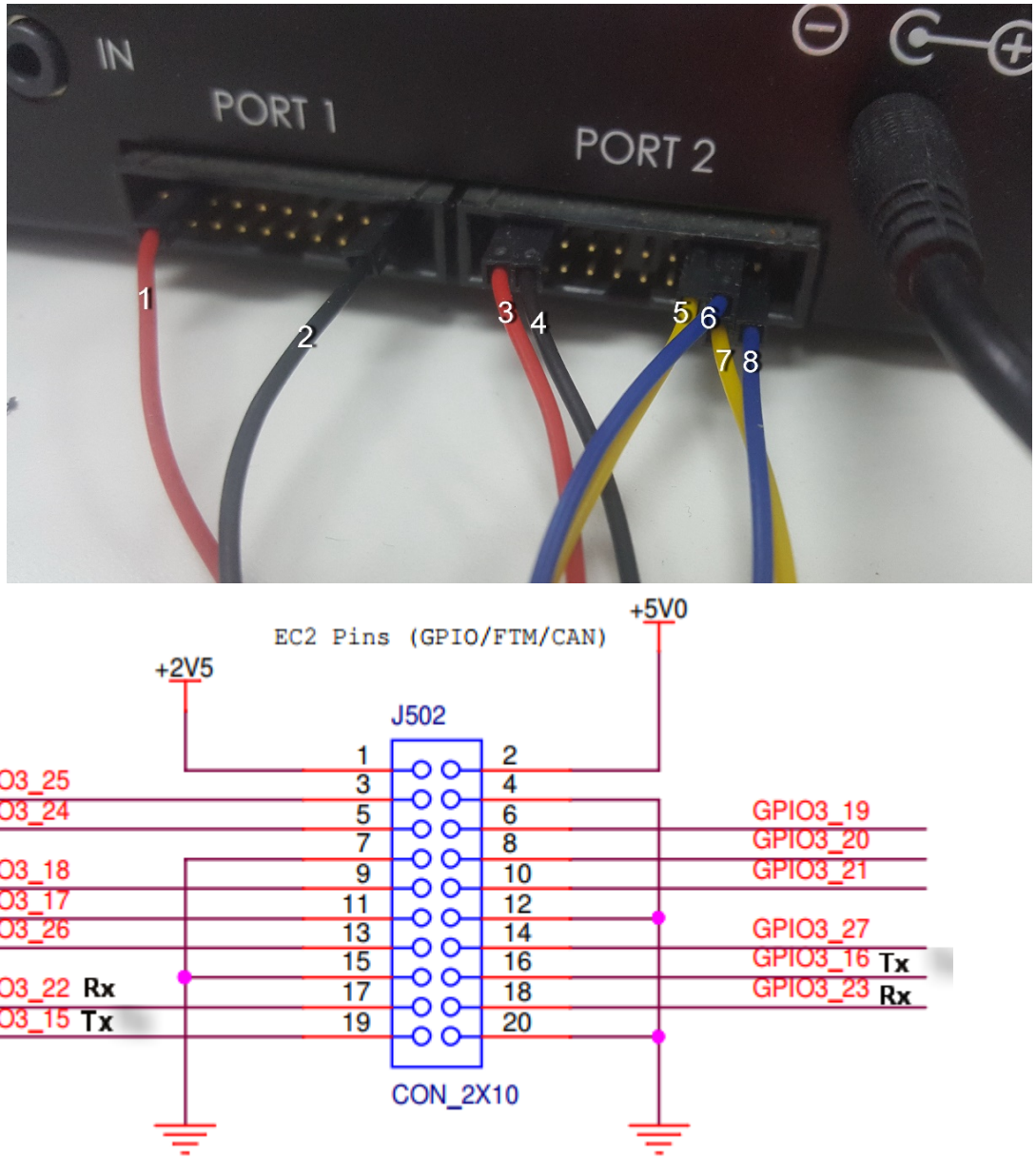


Figure 68. Hardware diagram for the FlexCan demo

NOTE

- Line1 and line3 are 5.0V.
- Line2 and line4 are GND.
- Line5 is CAN3 Tx.
- Line6 is CAN3 Rx.
- Line7 is CAN4 Rx.
- Line8 is CAN4 Tx.

11.3.2 Compiling the CANopen-app binary for the master node

CANopen application's name is CANopen-app. Perform the steps listed below to compile Canopen-app as linux command to the `target/usr/bin` directory.

1. Configure cross-toolchain on your host environment.
2. Use the commands below:

```
$ git clone ssh://git@bitbucket.sw.nxp.com/dnind/openil.git
$ cd openil
$ git checkout master
$ make nxp_ls1021aiot_baremetal_defconfig
$ make
```

3. The generated openil image file is in the `output/images/` directory.
4. Download the `sdcard.img` image file to the SD card:

In U-Boot mode, first run the `tftp` command for downloading sdcard.img to the buffer. Then, run the `mmc` command for downloading the sdcard.img to SD card.

NOTE

make sure the below options are enabled before building the image:

```
$ make menuconfig
Target packages --->
    Libraries --->
        Networking --->
              [*] canfestival
                    driver (socket) --->
                (--SDO_MAX_LENGTH_TRANSFER=512 --SDO_BLOCK_SIZE=75
                 --SDO_MAX_SIMULTANEOUS_TRANSFERS=1) additional
configure options
                [*] install examples
                [*] libsocketcan
     Networking applications --->
        [*] can-utils
        [*] iproute2
```

NOTE

- The following options are displayed only when the `canfestival` option is set to Y.
- Linux uses the SocketCAN interface, so the `driver`option selects the socket.
- The following `additional configure options` can be configured in the `config.h` file of CANopen:

  Parameter description:

  - --SDO_MAX_LENGTH_TRANSFER: Sets buffer size of SDO protocol.
  - -- SDO_BLOCK_SIZE: Sets the maximum number of frames that can be sent by SDO block transport protocol.
  - --SDO_MAX_SIMULTANEOUS_TRANSFERS: Sets the number of SDO modules.
- Install binary application to openil filesystem, if the`install examples` option is set to Y.

11.3.3 Running the CANopen application

First, boot the LS1021A-IoT board. Then, run the `CANopen-app` command in any directory in Linux prompt. While executing this command, first run the test code. After the test code is completed, you can implement the required instructions. The command `CANopen-app` execution process steps are described below:

1. First, indicate whether the CAN interface has opened successfully. All commands are dynamically registered. Then, indicate whether the command was registered successfully.

   - **Command registration log**

```
Command Registration Log:
[root@OpenIL:~]# CANopen-app
[   80.899975] IPv6: ADDRCONF(NETDEV_CHANGE): can0: link becomes ready
Note: open the CAN interface successfully!
"can_quit" command: register OK!
"setState" command: register OK!
"showPdo" command: register OK!
"requestPdo" command: register OK!
"sdo" command: register OK!
"" command: register OK!
"test_startM" command: register OK!
"test_sdoSingle" command: register OK!
"test_sdoSingleW" command: register OK!
"test_sdoBlock" command: register OK!
"test_showPdoCyc" command: register OK!
"test_showpdoreq" command: register OK!
"test_requestpdo" command: register OK!
```

2. There are nine test codes in total, tests1 to 9. Test code details are shown in the test log.

   - **Test code log** "---test---" indicates that the test code begins.
   - Firstly, the execution rights of the SDO and PDO protocol are explained.
   - The **tests 1~4** are SDO protocol test codes. After starting the CANopen master node, it automatically enters into initialization and pre-operation mode.
   - The **test5** is a test code that master node enters the operation mode and starts all slave nodes.
   - The **tests 6~9** are PDO protocol test codes.

```
Test Code Log:
------------------------- test ---------------------------
Note: Test code start execute...
      SDO protocol is valid in preoperation mode, but PDO protocol is invalid!
      SDO and PDO protocol are both valid in operation mode!
      Console is invalid when testing!
-----------------------------------------------------------
Note: test1--Read slave node single data by SDO.
Note: master node initialization is complete!
Note: master node entry into the preOperation mode!
Note: Alarm timer is running!
Note: slave node "0x02" entry into "Initialisation" state!
-----------------------------------------------------------
Note: test2--Write 0x2CD5 to slave node by SDO.
Note: Master write a data to 0x02 node successfully.
-----------------------------------------------------------
Note: test3--Read slave node single data by SDO again.
Note: reveived data is 0x2CD5
-----------------------------------------------------------
Note: test4--Read slave node block data by SDO.
---------------- text ------------------
Note: reveived string ==>
CANopen is a CAN-based communication system.
It comprises higher-layer protocols and profile specifications.
CANopen has been developed as a standardized embedded network with highly flexible
configuration capabilities.
```

```
It was designed originally for motion-oriented machine control systems, such as handling
systems.
Today it is used in various application fields, such as medical equipment, off-road vehicles,
maritime electronics, railway applications, or building automation.

---------------------------------------
------------------------------------------------------------
Note: test5--Master node entry operation mode, and start slave nodes!
Note: master node entry into the operation mode,and start all slave nodes!
------------------------------------------------------------
Note: test6--Master node show requested PDO data.
Note: Rpdo4 data is "       "
------------------------------------------------------------
Note: test7--Master node request PDO data.
------------------------------------------------------------
Note: test8--Master node show requested PDO data.
Note: Rpdo4 data is "require"
Note: slave node "0x02" entry into "Operational" state!
------------------------------------------------------------
Note: test9--Master node show received cycle PDO data.
Note: Rpdo2 data is "   cycle"
------------------------------------------------------------
```

NOTE

tests 1 to 9 are not commands.

3. After the test code is executed, it automatically prints the list of commands. Num00~06 are normal commands. After executing these instructions without parameters, the instruction usage is displayed. Num08~14 are test commands. All test commands except num10 have no parameters. Argument of Num10 is a 16-bit integer.
   - Now the user can execute any command in the command list.

   Command List

```
Command List:
------------------------------------------------------------------------------
 num    |       command        |                introduction
------------------------------------------------------------------------------
 00     | ctrl_quit            |  console thread exit!
------------------------------------------------------------------------------
 01     |  help                |  command list
------------------------------------------------------------------------------
 02     | can_quit           |  exit CANopen thread
------------------------------------------------------------------------------
 03     |  setState           |  set the CANopen node state
------------------------------------------------------------------------------
 04     |  showPdo           |  show the data of RPDO
------------------------------------------------------------------------------
 05     |  requestPdo       |  request the data of RPDO
------------------------------------------------------------------------------
 06     | sdo           |  read/write one entry by SDO protocol
------------------------------------------------------------------------------
 07     |                     |
------------------------------------------------------------------------------
 08     |  test_startM      |  test -- Start master
------------------------------------------------------------------------------
 09     |  test_sdoSingle      |  test -- Read slave node single data
------------------------------------------------------------------------------
 10     |  test_sdoSingleW     |  test -- Write slave node single data
------------------------------------------------------------------------------
```

```
 11    |  test_sdoBlock       |  test -- Read slave node block data
-------------------------------------------------------------------------
 12    |  test_showPdoCyc     |  test -- Show cycle PDO data
-------------------------------------------------------------------------
 13    |  test_showpdoreq     |  test -- Show requested PDO data
-------------------------------------------------------------------------
 14    |  test_requestpdo     |  test -- Request PDO data
-------------------------------------------------------------------------
Note: You can send command by console!
Note: Test code execution is complete!
```

Example: The following example shows the usage log after running the `sdo` command without any parameters.

```
SDO Command:
sdo
usage: sdo -type index subindex nodeid data
       type = "r"(read), "w"(write), "b"(block)
       index = 0~0xFFFF,unsigned short
       subindex = 0~0xFF,unsigned char
       nodeid = 1~127,unsigned char
       data = 0 ~ 0xFFFFFFFF
```

11.3.4 Running the Socketcan command

The following commands are executed on Linux. The standard Socketcan commands:

1. Open the can0 port.

```
$ ip link set can0 up
```

2. Close the can0 port.

```
$ ip link set can0 down
```

3. Set the baud rate to 500K for the can0 port

```
$ ip link set can0 type can bitrate 500000
```

4. Set can0 port to Loopback mode.

```
$ ip link set can0 type can loopback on
```

5. Send a message through can0. `002` (HEX) is node id, and this value must be 3 characters. `2288DD` (HEX) is a message, and can take a value up to 8 bytes.

```
$ cansend can0 002#2288DD
```

6. Monitor can0 port and wait for receiving data.

```
$ candump can0
```

7. See can0 port details.

```
$ ip -details link show can0
```

NOTE

The third and fourth commands are valid when the state of can0 port is closed.

Chapter 12
Known issues

The following table lists the known issues for Open Industrial Linux for this release.

Table 22. Known issues for this release

Item	Description
1	Need to define more relevant usage scenario for SAE AS6802 features (time-triggered, rate-constrained traffic). These are not used at the moment.
2	Sja1105-tool does not communicate with the BCM5464R PHY.

Chapter 13
Revision history

The table below summarizes revisions to this document.

Date	Document version	Topic cross- reference	Change description
15/10/2018	1.3.1	Getting Open IL	Updated the OpenIL version and Git tag
31/08/2018	1.3	EtherCAT	Added the chapter.
		FlexCAN	Added the chapter.
		i.MX6QSabreSD support.	Added the section in chapter: NXP OpenIL platforms. Updated other sections for i.MX6Q Sabre support.
		Getting Open IL	Updated the section.
		Selinux demo	Added the section, Installing basic packages on page 44 and updated Basic setup. Updates in other sections.
31/05/2018	1.2	Hardware requirements on page 30	Updated the section, RTnet hardware requirements.
		Software requirements on page 31	Updated the section, RTnet software requirements.
18/04/2018	1.1.1	RTnet on page 30	Added the section RTnet.
		Switch settings on page 22	Added a note for LS1043A switch setting.
30/03/2018	1.1	Supported industrial features on page 12	Added support for industrial IoT baremetal framework in this section.
		Booting up the board on page 16	Added a note for steps to be performed before booting up the board.
		Reference documentation on page 7	Added the section.
22/12/2017	1.0	OPC UA on page 75	Added the Chapter, 'OPC UA'.
		TSN Demo on page 82	Chapters for "1-board TSN demo" and "3-board TSN demo" replaced by a single chapter, "TSN demo".
		IEEE 1588 on page 33	• Updated the section, 'Industrial Features'. • -IEEE 1588 -'sja1105-ptp' support removed.
25/08/2017	0.3	-	Set up the OpenIL websitehttp://www.openil.org/.
		OTA implementation on page 127	OTA - Xenomai Cobalt 64-bit and SJA1105 support added.
		TSN Demo on page 82	Qbv support added.

Table continues on the next page...

Table continued from the previous page...

Date	Document version	Topic cross- reference	Change description
		SELinux on page 44	SELinux support for LS1043 / LS1046 Ubuntu Userland added.
		OP-TEE on page 39	OP-TEE support for LS1021ATSN platform added.
		4G-LTE Modem on page 125	4G LTE module - 64-bit support for LS1043ARDB, LS1046ARDB, and LS1012ARDB added.
		NXP OpenIL platforms on page 20	Ubuntu Userland support for 64-bit LS1043ARDB and 64-bit LS1046ARDB added.
26/05/2017	0.2	-	Initial public release.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

OpenIL_UG
Rev. 1.3.1
Oct/2018