NXP Semiconductors

User's Guide

Document identifier: OpenILUG

Rev. 1.7.1, 02/2020

Open Industrial 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
PDO	Process data object
PHC	PTP hardware clock

Table continues on the next page...

Table 1. Acronyms and abbreviations (continued)

Term	Description
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
   - LS1028ARDB Quick Start Guide
   - LX2160ARDB Quick Start Guide
2. For booting up LS1021A-TSN board, refer to the Section Booting up the board 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.

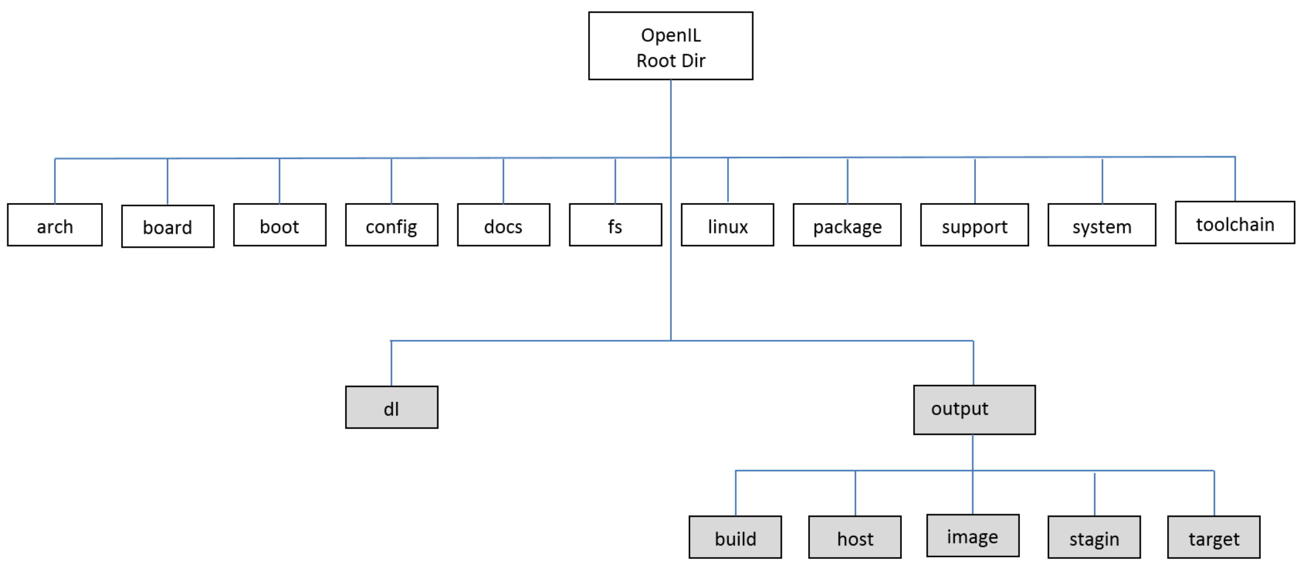


Figure 1. OpenIL structure

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
linux	The linux kernel package.

Table continues on the next page...

Table 2. Source directories (continued)

Directory name	Description
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
- Ubuntu 18.04

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	Remarks
which	

Table continues on the next page...

Table 4. Host system mandatory packages (continued)

Mandatory packages	Remarks
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)	
autogen autoconf libtool	
pkg-config	

Table 5. Host system optional packages

Optional packages	Remarks
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	Remarks
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 six NXP platforms that can support Ubuntu filesystem:

  — `configs/nxp_ls1043ardb-64b_ubuntu_defconfig`

  — `configs/nxp_ls1046ardb-64b_ubuntu_defconfig`

  — `configs/nxp_ls1028ardb-64b_ubuntu_defconfig`

- `configs/nxp_ls1021aiot_ubuntu_defconfig`
- `configs/imx6q-sabresd_ubuntu_defconfig`
- `configs/nxp_lx2160ardb-64b_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
- FlexCan
- EtherCAT
- NFC-Clickboard
- BEE-Clickboard
- BLE-Clickboard

These are explained in detail in Industrial features.

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

Table continues on the next page...

Table 6. Supported NXP platforms (continued)

Platform	Architecture	Configuration file in OpenIL	Boot
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
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
ls1028ardb (64-bit)	ARM v8	configs/nxp_ls1028ardb-64b_defconfig	SD
ls1028ardb (Ubuntu)	ARM v8	configs/nxp_ls1028ardb-64b_ubuntu_defconfig	SD
lx2160ardb (64-bit)	ARM v8	configs/nxp_lx2160ardb-64b_defconfig	SD
lx2160ardb (Ubuntu)	ARM v8	configs/nxp_lx2160ardb-64b_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-18.04.3-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-18.04.3-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
ls1028ardb	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
ls1028ardb	gcc 5.x	glibc 2.23	Systemd	ubuntu-base-18.04.3-arm64

Table continues on the next page...

Table 7. Default compilation settings (continued)

Platform	Toolchain	libc	Init system	Filesystem
lx2160ardb	gcc 5.x	glibc 2.23	BusyBox	OpenIL default
lx2160ardb	gcc 5.x	glibc 2.23	Systemd	ubuntu-base-18.04.3-arm64

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 2020.01 v1.7 release
$ git checkout OpenIL-v1.7-202001 -b OpenIL-v1.7-202001
```

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

Table continues on the next page...

Table 8. Default configuration (continued)

Platform	Configuration file in OpenIL
ls1021atsn (OP-TEE)	configs/nxp_ls1021atsn_optee-sb_defconfig
ls1021aiot	configs/nxp_ls1021aiot_defconfig
ls1021aiot (OP-TEE)	configs/nxp_ls1021aiot_optee_defconfig
ls1028ardb (64-bit)	configs/nxp_ls1028ardb-64b_defconfig
ls1028ardb (Ubuntu)	configs/nxp_ls1028ardb-64b_ubuntu_defconfig
ls1028ardb (64bit)	configs/nxp_ls1028ardb-64b-emmc_defconfig
ls1028ardb (64bit)	configs/nxp_ls1028ardb-64b-xspi_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 (64-bit)	configs/nxp_ls1046ardb-64b-emmcboot_defconfig
ls1046ardb (64-bit)	configs/nxp_ls1046ardb-64b_qspi_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
lx2160ardb (64bot)	configs/nxp_lx2160ardb-64b_defconfig
lx2160ardb (64bit)	configs/nxp_lx2160ardb-64b-xspi_defconfig
lx2160ardb (Ubuntu)	configs/nxp_lx2160ardb-64b_ubuntu_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, LS1046ARDB), run the following commands:

```
$ cd openil
$ make nxp_ls1046ardb-64b_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 the BL2, BL31, BL33 binary from ATF;
- Creates a root filesystem in selected formats.
- Generates the Image file for booting;

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

```
images/
├── bl2_rcw.pbl                 --- BL2 + RCW
├── fip_uboot.bin               --- BL31 + BL33 (uboot)
├── rcw_1800_sdboot.bin         --- RCW binary
├── boot.vfat
├── fsl-ls1046a-rdb-sdk.dtb     --- dtb file for ls1046ardb
├── 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-dtb.bin              --- uboot image for ls1046ardb
└── Image                       --- kernel image for ls1046ardb
```

Notice: Image file name used for each configurations as following described:

- xspi.cpio.img: *xspi_defconfig
- sdcard.img: default and *emmc_defconfig
- qspi.cpio.img: *qspi_defconfig

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.

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 NXP boards

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
LS1028ARDB	SD card	sdcard.img	SW2[1-8] = 0b'10001000
LX2160ARDB	SD card	sdcard.img	SW1[1-4] = 0b'1000
i.MX6Q SabreSD	SD card	sdcard.img	SW6 = 0b'01000010

The flash image (*sdcard.img or qspi.img*) includes all the information: 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/FlexSPI 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.

FlexSPI (XSPI, image name is xspi.cpio.img) boot has the same commands to make the flash.

```
=>i2c mw 0x24 0x7 0xfc; i2c mw 0x24 0x3 0xf5
=>tftp 0x80000000 qspi.cpio.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.

In OpenIL, all packages used are in directory "`./package/`", and the package name is the sub-directory name. Linux kernel and uboot are also packages, the package name for Linux kernel is `linux`, and package name for u-boot is `uboot`.

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
<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, LX2160ARDB 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-64b_defconfig
# or
$ make nxp_ls1046ardb-64b_defconfig
$ make
# or make with a log
$ make 2>&1 | tee build.log
```

- Programming BL2, RCW, BL31, U-Boot and FMan ucode in SD card

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

```
# programming BL2 and RCW
=> tftpboot 82000000 bl2_rcw.pbl
=> mmc erase 8 800
=> mmc write 82000000 8 800
# programming the FMan ucode
=> tftpboot 82000000 fmucode.bin
=> mmc erase 0x4800 0x200
=> mmc write 82000000 0x4800 0x200
# programming the BL31 and U-Boot firmware
=> mmc erase 0x800 0x2000
=> tftpboot 82000000 fip_uboot.bin
=> mmc write 82000000 0x800 0x2000
#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 83000000 Image
=>tftp 88000000 rootfs.ext2.gz.uboot
=>tftp 8f000000 fsl-ls1046a-rdb-sdk.dtb
# or for ls1043ardb
=>tftp 8f000000 fsl-ls1043a-rdb-sdk.dtb
=>booti 83000000 88000000 8f000000
```

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-64b_defconfig
$ make
# or make with a log
$ make 2>&1 | tee build.log
```

- Programming BL2, BL31, U-Boot, RCW and pfe firmware in QSPI

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

```
 # programming BL31 and U-Boot
=>i2c mw 0x24 0x7 0xfc; i2c mw 0x24 0x3 0xf5
=>tftp 0x80000000 fip_uboot.bin
=>sf probe 0:0
=>sf erase 0x100000 +$filesize
=>sf write 0x80000000 0x100000 $filesize
 # programming BL2 and RCW
=>i2c mw 0x24 0x7 0xfc; i2c mw 0x24 0x3 0xf5
=>tftp 0x80000000 bl2_rcw.pbl
=>sf probe 0:0
=>sf erase 0x0 +$filesize
=>sf write 0x80000000 0x0 $filesize
 # programming pfe firmware
=> tftp 0x80000000 pfe_fw_sbl.itb
=> sf probe 0:0
=> sf erase 0xa00000 +$filesize
=> sf write 0x80000000 0xa00000 $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.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.

3.7 LS1028ARDB

The QorIQ® LS1028A reference design board (LS1028ARDB) is a computing, evaluation, development, and test platform supporting the QorIQ LS1028A processor, which is a dual-core Arm® Cortex®-v8 A72 processor with frequency up to 1.3 GHz. The LS1028ARDB is optimized to support SGMII (1 Gbit/s), QSGMII (5 Gbit/s), PCIe x1 (8 Gbit/s), and SATA (6 Gbit/s) over high-speed SerDes ports, USB 3.0, DisplayPort, and also a high-bandwidth DDR4 memory. The LS1028ARDB can be used to develop and demonstrate human machine interface systems, industrial control systems such as robotics controllers and motion controllers, and PLCs. The reference design also provides the functionality needed for Industrial IoT gateways, edge computing, industrial PCs, and wireless or wired networking gateways.

3.7.1 Switch settings

The following table lists and describes the switch configuration for LS1028ARDB board.

Platform	Boot source	SW setting
LS1028ARDB	SD	sw2: 0b’10001000

3.7.2 Interface naming

The following section ddescribes the association between physical interfaces and networking interfaces as presented by the software.

3.7.2.1 Interface naming in U-Boot

The following figure shows the Ethernet ports as presented in U-Boot:

Note: In U-Boot running on RDB, only *enetc#0* is functional.

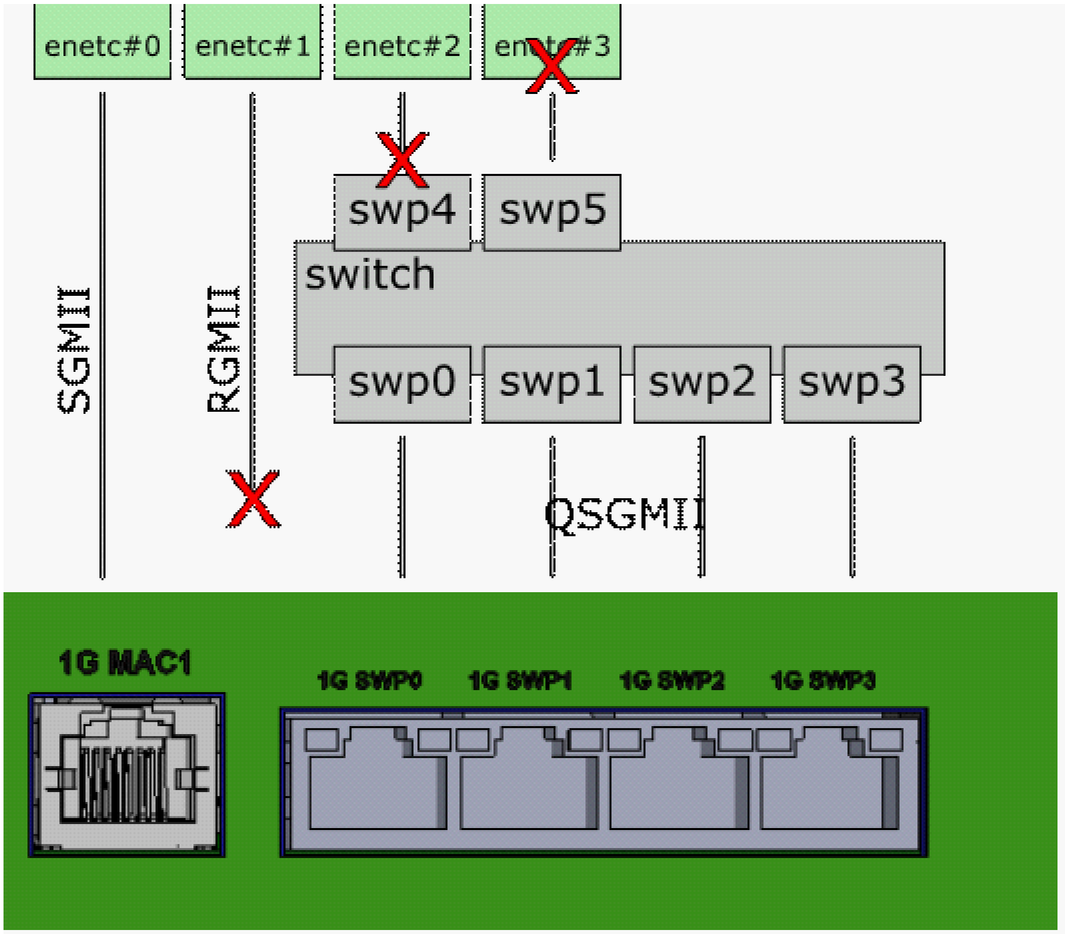


Figure 3. Ethernet ports in U-Boot

Table 16. Interface naming in U-Boot

RDB port	U-Boot interface	PCI function	Comments
1G MAC1	*enetc#0*	0000:00:00.0	`enetc#0` is 1G SGMII port of ENETC.
N/A	*enetc#1*	0000:00:00.1	*enetc#1* is presented in U-Boot on all boards. This interface is not functional on RDB.
Internal	*enetc#2*	0000:00:00.2	Connected internally (MAC to MAC) to the Ethernet switch. Note that the switch is not initialized in U-Boot; therefore, this interface is not functional.
Internal	*enetc#3*	0000:00:00.6	Connected internally (MAC to MAC) to the Ethernet switch. This interface is presented if bit 851 is set in RCW. Note that the switch is not initialized in u-boot; therefore, this interface is not functional.
1G SWP0 to 1G SWP3	N/A	0000:00:00.5	The switch is currently not initialized by U-Boot; therefore, these interfaces are not functional.

3.7.2.2 Interface naming in Linux

The following figure shows how Ethernet ports are presented in Linux.

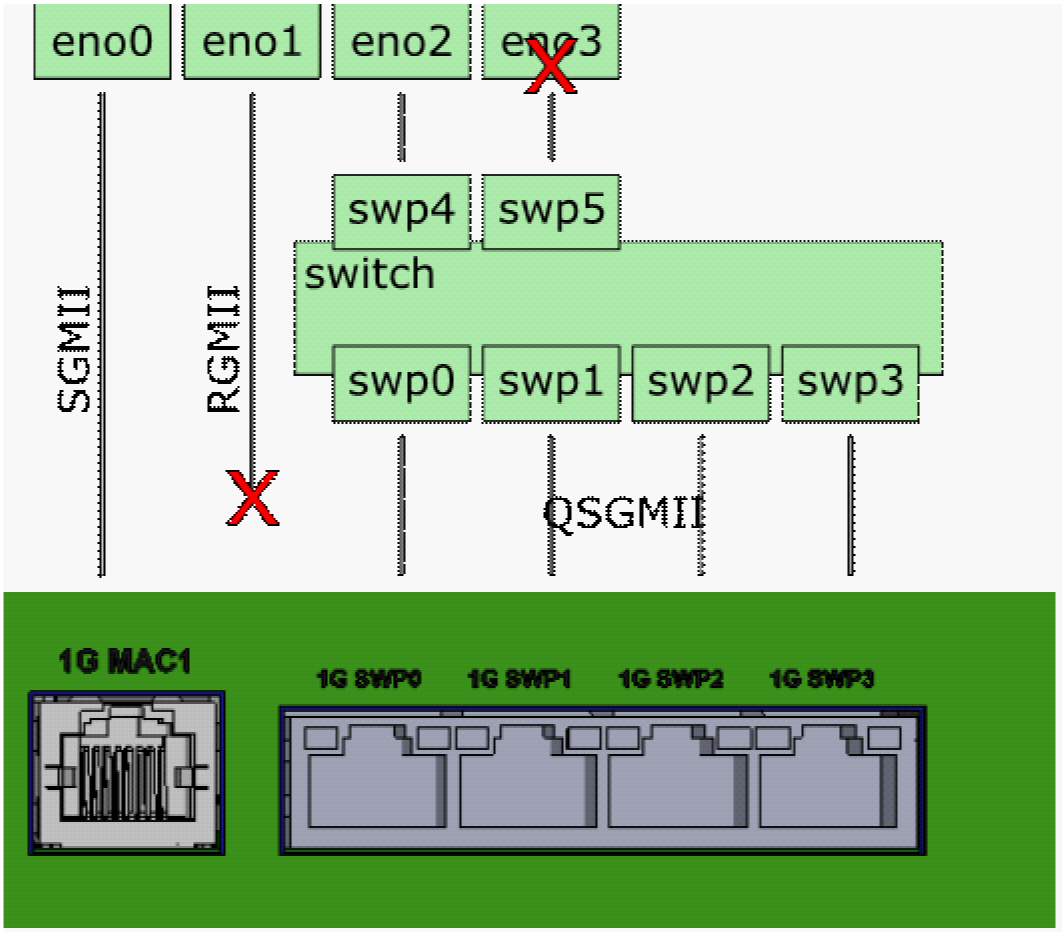


Figure 4. Ethernet ports in Linux

Table 17. Interface naming in Linux

RDB port	Linux netdev	PCI function	Comments
1G MAC1	*eno0*	0000:00:00.0	

N/A	*eno1*	0000:00:00.1	RGMII interface is not present on RDB board and the associated ENETC interface is disabled in device tree:
			`&enetc_port1`
			`{ status = "disabled";`
			`}`

Internal	*eno2*	0000:00:00.2	Connected internally (MAC to MAC) to *swp4*. This is used to carry traffic between the switch and software running on ARM cores.

Table continues on the next page...

Table 17. Interface naming in Linux (continued)

Internal	*eno3*	0000:00:00.6	Connected internally (MAC to MAC) to *swp5*. This is intended to be used by user- space data-path applications and is disabled by default. It can be enabled by setting bit 851 in RCW.
1G SWP0 to 1G SWP3	*swp0* to *swp3*	0000:00:00.5	By default, switching is not enabled on these ports. For detail on how to enable switching across these ports, see Felix Ethernet switch on page 78.
Internal	*swp4*		Connected internally (MAC to MAC) to *eno2*.
Internal	*swp5*		Last switch port (connected to *eno3*) is currently not presented in Linux.

3.7.3 Updating target images

This section describes how to update the target images for NXP's LS1028ARDB platforms. For this platform, OpenIL can support 64-bit systems. Use the following commands to build the images for the LS1028ARDB platforms:

1. **Building images**

```
$ cd openil
$ make  nxp_ls1028ardb-64b_defconfig
$ make  # or make with a log
$ make 2>&1 | tee build.log
```

2. **Programming BL2, RCW, BL31, U-Boot in SD card**:

   Power on the LS1028ARDB board to U-Boot command environment, then use the following commands:

```
# programming the BL2 and RCW binary
     => tftpboot 82000000 bl2_rcw.pbl
     => mmc erase 8 0x800
     => mmc write 0x82000000 8 0x800
# programming BL31 and U-Boot
     => tftpboot 82000000 fip_uboot.bin
     => mmc erase 0x800 0x800
     => mmc write 82000000 0x800 0x2000
# programming the u-boot environment
     => tftpboot 82000000 uboot-env.bin
     => mmc erase 0x2800 0x800
     => mmc write 82000000 0x2800 0x800
#then reset the board
```

3. **Deploying kernel and Ramdisk from TFTP**

   - Set the U-Boot environment using the commands below:

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

   - Boot up the system

```
=> tftp 83000000 Image
=> tftp 88000000 rootfs.ext2.gz.uboot
```

```
=> tftp 8f000000 fsl-ls1028a-rdb.dtb
=> booti 83000000 88000000 8f000000
```

3.8 LX2160ARDB

The QorIQ LX2160A processor is built on NXP's software-aware, core-agnostic DPAA2architecture, which delivers scalable acceleration elements sized for application needs,unprecedented efficiency, and smarter, more capable networks. When coupled with easeof-use facilities such as real-time monitoring and debug, virtualization, and softwaremanagement utilities, the available toolkits allow for both hardware and softwareengineers to bring a complete solution to market faster than ever.

The LX2160A integrated multicore processor combines sixteen Arm® Cortex®-A72processor cores with 24 lanes of the latest 25 GHz SerDes technology supporting highperformance Ethernet speeds (10 Gbps, 25 Gbps, 40 Gbps, 50 Gbps, and 100 Gbps) andPCI express to Gen4 (16 Gbps). With the low power of FinFET process technology andcommon network and peripheral bus interfaces, the LX2160A is well suited fornetworking, telecom/datacom, wireless infrastructure, storage and military/ aerospaceapplications..

The LX2160A processor is supported by a consistent API that provides both basic andcomplex manipulation of the hardware peripherals in the device, releasing the developerfrom the classic programming challenges of interfacing with new peripherals at thehardware level.

The QorIQ LX2160A reference design board is a 1U form-factor tool for evaluation and design of value-added networking applications such as 5G packet processing, network-function virtualization (NFV) solutions, edge computing, white box switching, industrial applications, and storage controllers.

3.8.1 Switch settings

The following table lists and describes the switch configuration for LX2160ARDB board.

Platform	Boot source	SW setting
LX2160ARDB	SD	sw1[1~4]: 0b’1000

3.8.2 Updating target images

Use the following commands to build the images for LX2160ARDB platform:

- **Building images**

```
$ cd openil
$ make nxp_lx2160ardb-64b_defconfig
$ make
# or make with a log
$ make 2>&1 | tee build.log
```

- **Programming BL2 and RCW , BL31 and U-Boot on the SD card**

  Power on the LX2160ARDB board to U-Boot command environment. Then, use the commands below:

```
# flash BL2 and RCW binary
=>tftp 81000000 bl2_rcw.pbl
=>mmc erase 8 0x500
=>mmc write 0x81000000 8 0x500
# flash BL31 and U-Boot binary
=>tftp 81000000 fip_uboot.bin
=>mmc erase 0x800 0x2000
=>mmc write 0x81000000 0x800 0x2000
# flash DDR firmware
```

```
=>tftp 81000000 fip_ddr.bin
=>mmc erase 0x4000 0x400
=>mmc write 0x81000000 0x4000 0x400
# flash phy-ucode firmware
=>tftp 81000000 phy-ucode.txt
=>mmc erase 0x4C00 0x200
=>mmc write 0x81000000 0x4C00 0x200
# flash MC firmware
=>tftp 81000000 mc.itb
=>mmc erase 0x5000 0x1800
=>mmc write 0x81000000 0x5000 0x1800
# flash  dpl-eth firmware
=>tftp 81000000 dpl-eth.19.dtb
=>mmc erase 0x6800 0x800
=>mmc write 0x81000000 0x6800 0x800
# flash dpc-usxgmii firmware
=>tftp 81000000 dpc-usxgmii.dtb
=>mmc erase 0x7000 0x800
=>mmc write 0x81000000 0x7000 0x800
#then reset the board
```

- Deploying kernel and Ramdisk from TFTP
  1. Set the U-Boot environment.

```
=>setenv bootargs 'console=ttyAMA0,115200 root=/dev/ram0 rw rootwait
earlycon=pl011,mmio32,0x21c0000'
=>saveenv
```

  2. Boot up the system.

```
=>mmcinfo;mmc read $mc_fw_addr 0x05000 0x1800;mmc read $dpc_addr 0x07000 0x800;mmc read
$dpl_addr 0x06800 0x800;fsl_mc start mc $mc_fw_addr $dpc_addr;fsl_mc apply dpl $dpl_addr;
=>tftp 83000000 Image
=>tftp 88000000 rootfs.ext2.gz.uboot
=>tftp 8f000000 fsl-lx2160a-rdb.dtb
=>booti 83000000 88000000 8f000000
```

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

For the Industrial IoT baremetal framework, refer to the document, *Industrial_IoT_Baremetal_Framework_Developer_Guide* available at https://www.nxp.com/support/developer-resources/nxp-designs/time-sensitive-networking-solution-for-industrial-iot:LS1021A-TSN-RD?tab=Documentation_Tab.

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, LS1028ARDB, 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 (Just for Arm V7 SoC):

  ```
  $ 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).
- For LS1028A, two LS1028ARDB 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 LS1043ARDB or LS1046ARDB board.

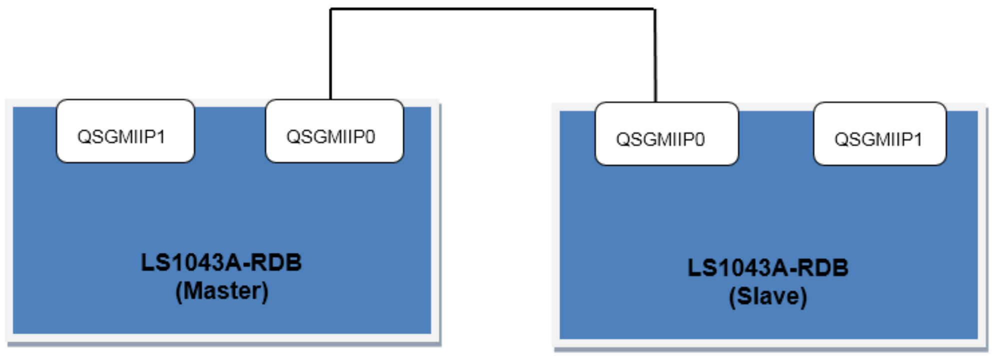


Figure 5. Hardware setup for RTnet (LS1043A as an example)

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
```

2. Alternatively, for configuring LS1046ARDB in the `openil` directory, use the command below:

```
make nxp_ls1046ardb-64b_defconfig
```

3. Or, for configuring LS1028ARDB in the openil directory, use the command below:

```
make nxp_ls1028ardb-64b_defconfig
```

4. 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)
```

For ENETC devices

- Disable the Linux ENETC driver using the settings below:

```
$make linux-menuconfig
      Device Drivers --->
        Network device support  --->
                  Ethernet   driver support  --->
                       < >     ENETC PF driver
                       < >     FELIX switch driver
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> ENETC
```

5. Now, run the `make` command to build all images.
6. After flashing images to the SD card, boot LS1043ARDB or LS1046ARDB from the SD card and enter the Linux prompt.
7. 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”.

c. ENETC devices

- Master board
    - — RT_DRIVER= "rt_enetc” - The driver used (currently, it is 'rt_enetc').
    - — 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_enetc” - The driver used (currently, it is 'rt_enetc').
    - — 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 18. 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.

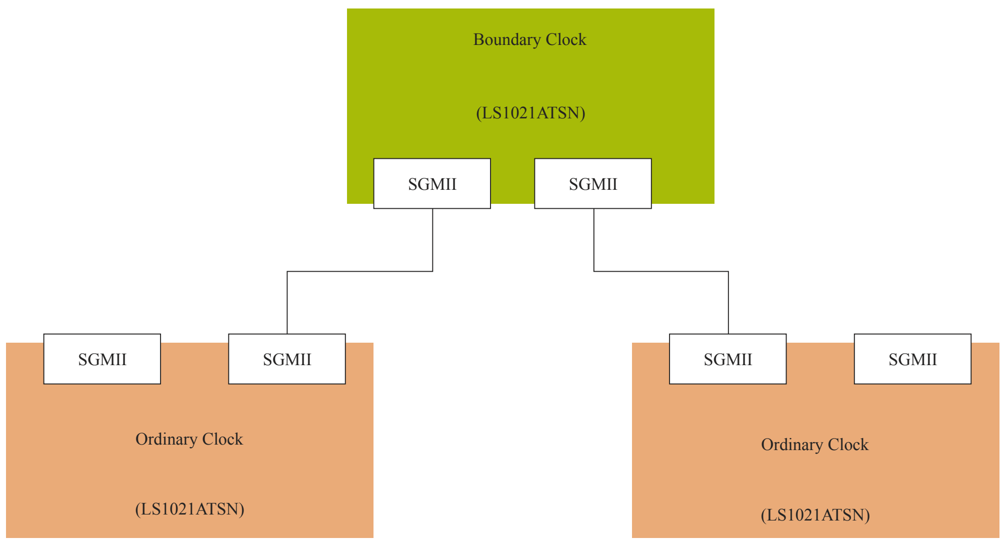


Figure 6. LS1021ATSN BC synchronization

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 19. 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.

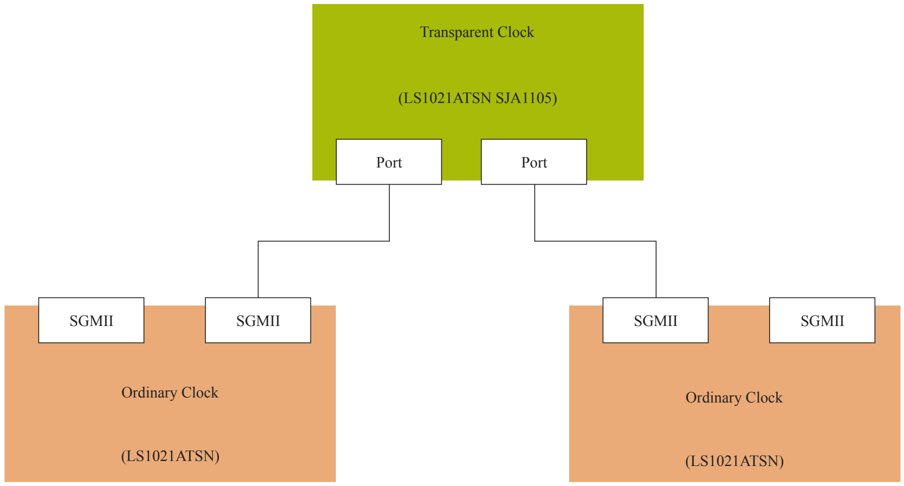


Figure 7. LS1021ATSN TC synchronization

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

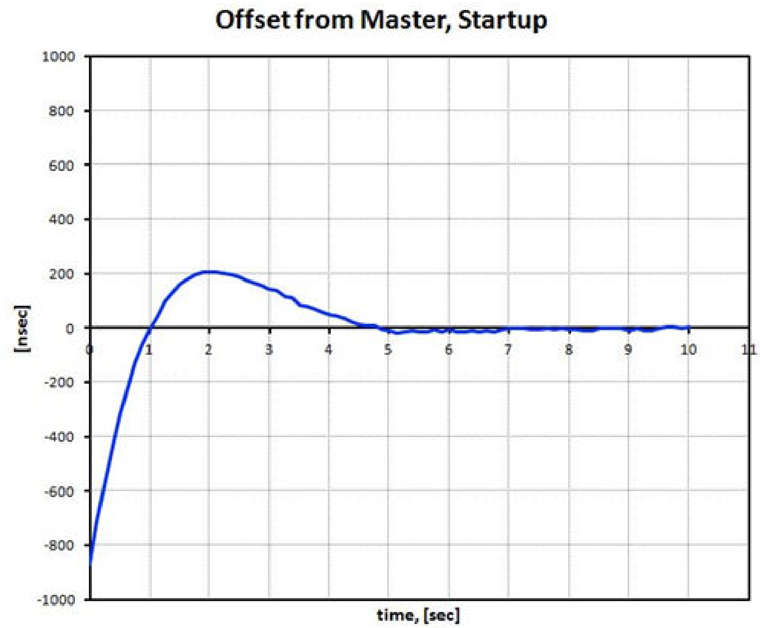


Figure 8. Offset from master in start up state

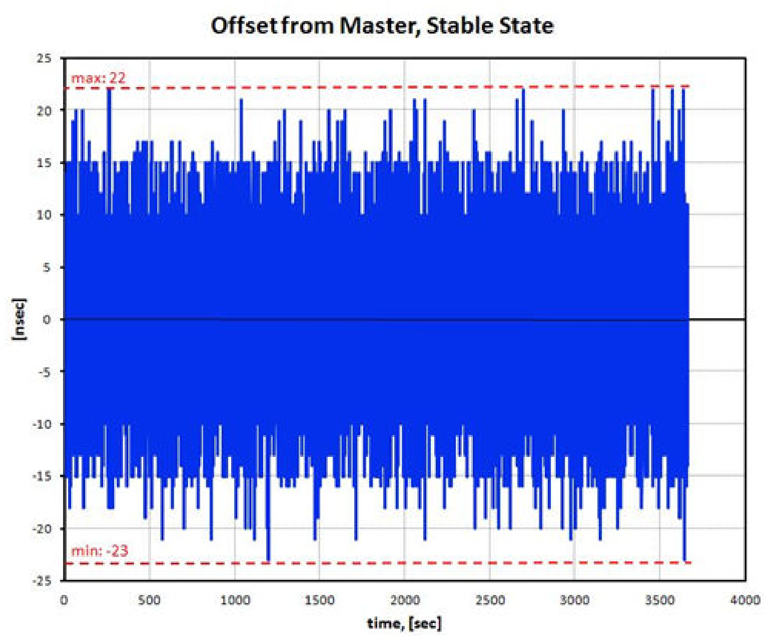


Figure 9. Offset from master in stable state

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.

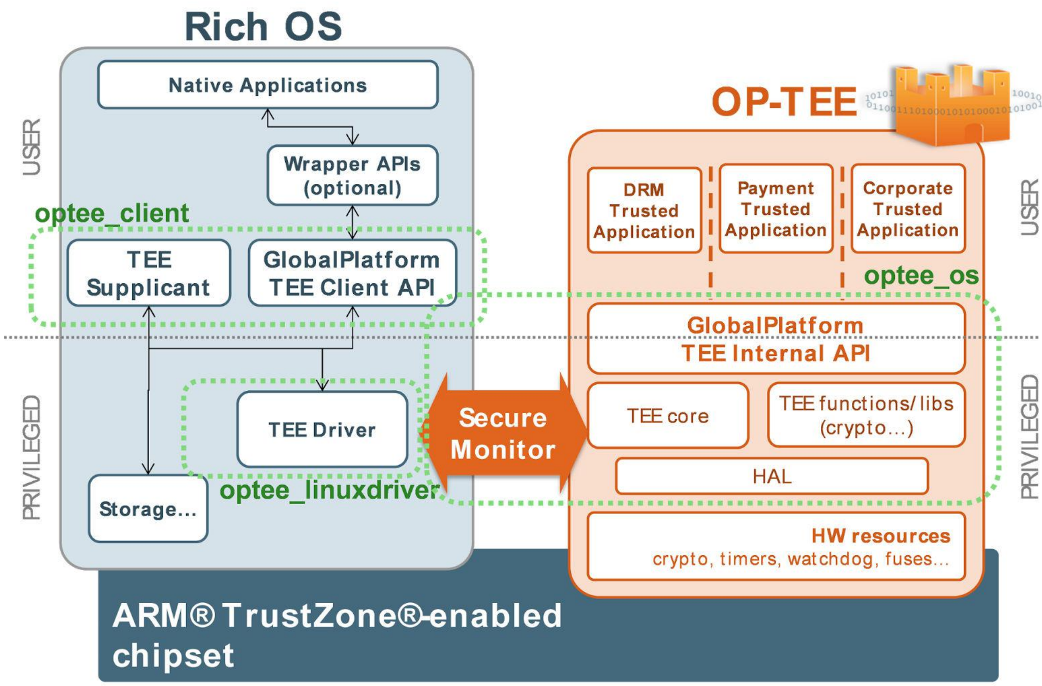


Figure 10. Architecture of OP-TEE on an ARM TrustZone enabled SoC

4.5.3 DDR memory map

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

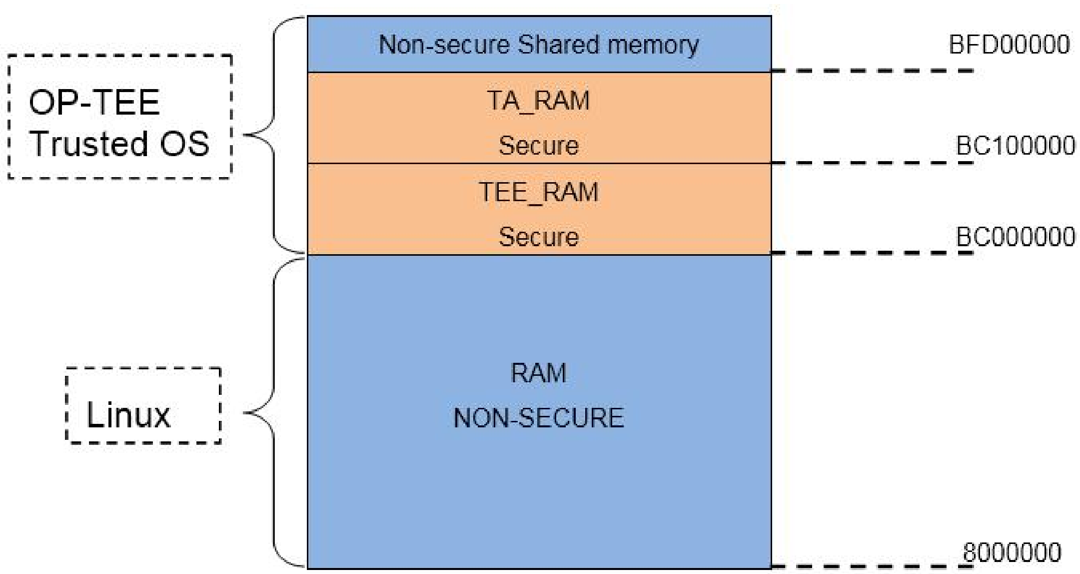


Figure 11. DDR memory map

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 the NXP platforms:- LS1028ARDB, 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 # for ls1043ardb-64b platform
# or
$ make nxp_ls1046ardb-64b_ubuntu_defconfig # for ls1046ardb-64b platform
# or
$ make nxp_ls1028ardb-64b_ubuntu_defconfig # for ls1028ardb-64b platform

$ 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.

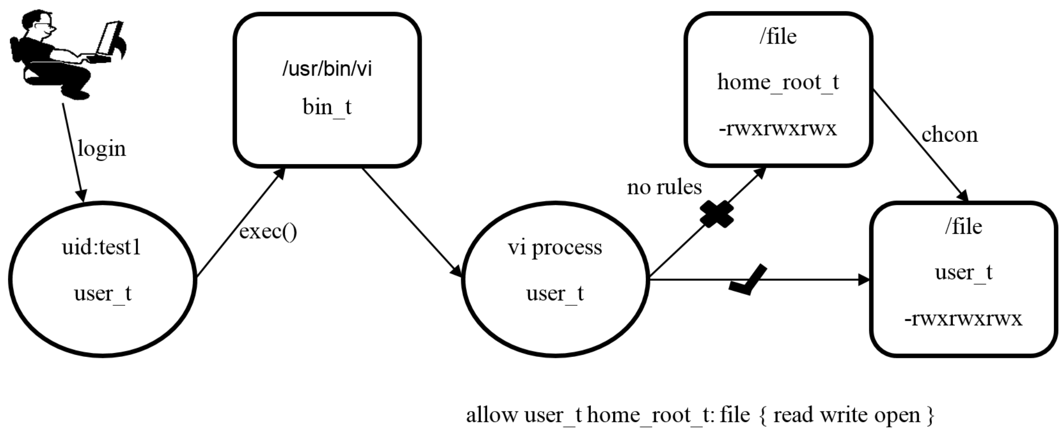


Figure 12. Allowing local file access control

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 13. 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 14. 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 15. 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.

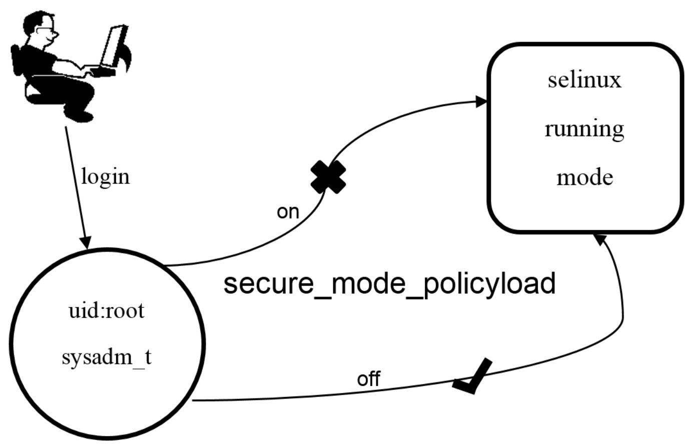


Figure 16. Restricting root permissions

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.

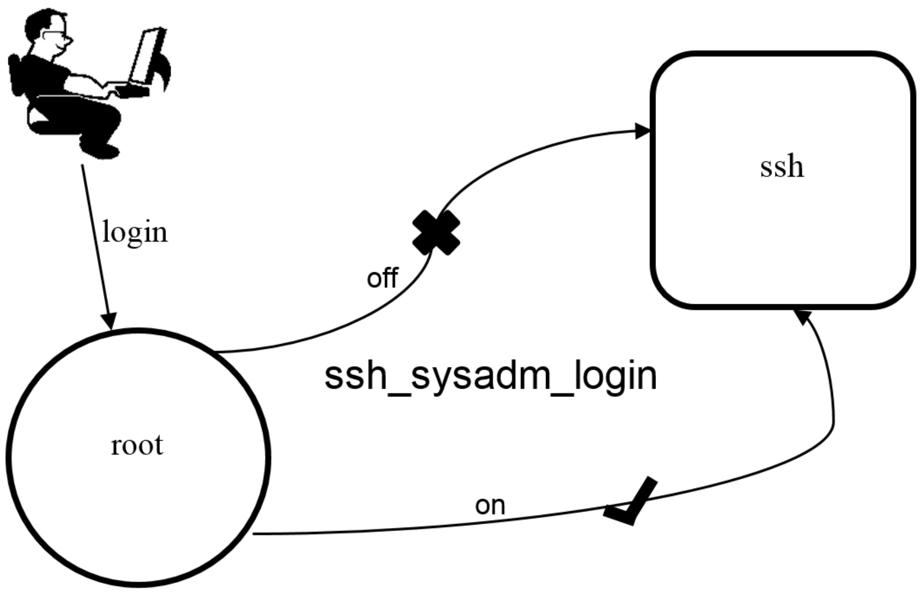


Figure 17. ssh remote permission

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
```

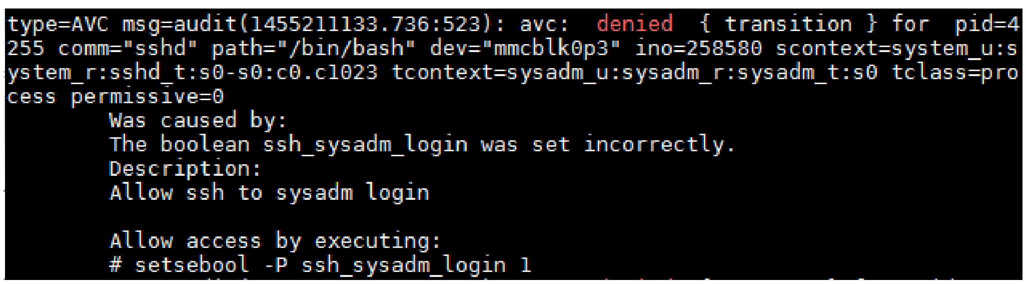


Figure 18. Audit log for sshd

```
$ audit2allow -a
```

Chapter 5
IEEE 1588/802.1AS

IEEE 1588 is the IEEE standard for a precision clock synchronization protocol for networked measurement and control systems.

IEEE 802.1AS is the IEEE standard for local and metropolitan area networks – timing and synchronization for time-sensitive applications in bridged local area networks. It specifies the use of IEEE 1588 specifications where applicable in the context of IEEE Std 802.1D-2004 and IEEE Std 802.1Q-2005.

5.1 Introduction

NXP's QorIQ platform provides hardware assist for 1588 compliant time stamping with the 1588 timer module. The software components required to run IEEE 1588/802.1AS protocol utilizing the hardware feature are listed below:

1. Linux PTP Hardware Clock (PHC) driver

2. Linux Ethernet controller driver with hardware timestamping support

3. A software stack application for IEEE 1588/802.1AS

NOTE

In this document, IEEE 1588 mentioned is IEEE 1588-2008, and IEEE 802.1AS mentioned is IEEE 802.1AS-2011.

5.2 Device types

There are five basic types of PTP devices in IEEE 1588.

- 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.)

5.3 Two types of time-aware systems in IEEE 802.1AS

In gPTP, there are only two types of time-aware systems: end stations and Bridges, while IEEE 1588 has ordinary clocks, boundary clocks, end-to-end transparent clocks, and P2P transparent clocks. A time-aware end station corresponds to an IEEE

1588 ordinary clock, and a time-aware Bridge is a type of IEEE 1588 boundary clock where its operation is very tightly defined, so much so that a time-aware Bridge with Ethernet ports can be shown to be mathematically equivalent to a P2P transparent clock in terms of how synchronization is performed.

1. Time-aware end station

An end station that is capable of acting as the source of synchronized time on

the network, or destination of synchronized time using the IEEE 802.1AS protocol, or both.

2. Time-aware bridge

A Bridge that is capable of communicating synchronized time received on one

port to other ports, using the IEEE 802.1AS protocol.

5.4 linuxptp stack

Features of open source linuxptp

- 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 clock_adjtimex system call.
- Implements Boundary Clock (BC), Ordinary Clock (OC) and Transparent Clock (TC).
- Transport over UDP/IPv4, UDP/IPv6, and raw Ethernet (Layer 2).
- Supports IEEE 802.1AS-2011 in the role of end station.
- Modular design allowing painless addition of new transports and clock servos.
- Implements unicast operation.
- Supports a number of profiles, including:
  - The automotive profile.
  - The default 1588 profile.
  - The enterprise profile.
  - The telecom profiles G.8265.1, G.8275.1, and G.8275.2.
  - Supports the NetSync Monitor protocol.
- Implements Peer to peer one-step.
- Supports bonded, IPoIB, and vlan interfaces.

Features added by OpenIL

- Supports IEEE 802.1AS-2011 in the role of time-aware bridge.
- Supports synchronization to LS1021ATSN SJA1105 switch with sja1105-tool APIs.

5.5 Quick Start for IEEE 1588

5.5.1 Ordinary clock verification

Connect two network interfaces in back-to-back manner for two boards. Make sure there is no MAC address conflict on the boards, the IP addresses are set properly and ping the test network. Run `linuxptp` on each board. For example, `eth0` is used on each board.

```
$ ptp4l -i eth0 -m
```

On running the above command time synchronization will start, and the slave linuxptp selected automatically will synchronize to master with synchronization messages displayed, such as time offset, path delay and so on.

5.5.2 Boundary clock verification

At least three boards are needed. Below is an example for three boards network connection. Make sure there is no MAC address conflict on the boards, the IP addresses are set properly and ping the test network.

```
Board1---eth0------------------Board2 eth0
|
|
--eth1------------------Board3 eth0
```

Run `linuxptp` on Board1 (boundary clock).

```
$ ptp4l -i eth0 -i eth1 -m
```

Run `linuxptp` on Board2/Board3 (ordinary clock).

```
$ ptp4l -i eth0 -m
```

On running the above command, time synchronization will start, and the slaves linuxptp selected automatically will synchronize to the unique master with synchronization messages displayed such as time offset, path delay and so on.

5.5.3 Transparent clock verification

At least three boards are needed. Below is an example for three boards network connection. Make sure there is no MAC address conflict on the boards, the IP addresses are set properly, and ping the test network.

```
Board1---eth0------------------Board2 eth0
|
|
--eth1------------------Board3 eth0
```

Run `linuxptp` on Board1 (transparent clock). If want Board1 works as E2E TC, use E2E-TC.cfg. If want Board1 works as P2P TC, use P2P-TC.cfg.

```
$ ptp4l -i eth0 -i eth1 -f /etc/ptp4l_cfg/E2E-TC.cfg -m
```

Run `linuxptp` on Board2/Board3 (ordinary clock).

```
$ ptp4l -i eth0 -m
```

On running the above commands, time synchronization will start between ordinary clocks, and the slave linuxptp selected automatically will synchronize to the master with synchronization messages displayed such as time offset, path delay and so on.

5.6 Quick Start for IEEE 802.1AS

The following sections describe the steps for implementing IEEE 802.1AS on NXP boards.

5.6.1 Time-aware end station verification

Connect two network interfaces in back-to-back way for two boards. Make sure no MAC address conflict on the boards, IP address set properly and ping test work.

Remove below option in /etc/ptp4l_cfg/gPTP.cfg to use default larger value, because estimate path delay including PHY delay may exceed 800ns since hardware is using MAC timestamping.

```
neighborPropDelayThresh 800
```

Run linuxptp on each board. For example, eth0 is used on each board.

```
$ ptp4l -i eth0 -f /etc/ptp4l_cfg/gPTP.cfg -m
```

Time synchronization will start, and the slave linuxptp selected automatically will synchronize to master with synchronization messages printed, like time offset, path delay and so on.

5.6.2 Time-aware bridge verification

At least three boards are needed. Below is an example for three boards network connection. Make sure no MAC address conflict on the boards, IP address set properly and ping test work.

```
Board1---eth0-----------------Board2 eth0
```

|

|

```
--eth1-----------------Board3 eth0
```

Remove below option in /etc/ptp4l_cfg/gPTP.cfg to use default larger value, because estimate path delay including PHY delay may exceed 800ns since hardware is using MAC timestamping.

```
neighborPropDelayThresh 800
```

Run linuxptp on Board1 (time-aware bridge).

```
$ ptp4l -i eth0 -i eth1 -f /etc/ptp4l_cfg/gPTP.cfg -m
```

Run linuxptp on Board2/Board3 (time-aware end station).

```
$ ptp4l -i eth0 -m
```

Time synchronization will start between three boards, and the slaves linuxptp selected automatically will synchronize to the unique master with synchronization messages printed, like time offset, path delay and so on.

5.7 Known issues and limitations

1. When LS1028A TSN switch in Linux is configured as L2 switch, the interfaces should not be configured with IP addresses. Running linuxptp on these interfaces must use Ethernet protocol instead of UDP/IP. The method is to add an option “-2” executing ptp4l command. For example,

```
$ ptp4l -i eth0 -2 -m
```

5.8 Long term test

No changes.

Chapter 6
NETCONF/YANG

This chapter provides an overview of the NETCONF protocol and Yang (a data modelling language for NETCONF). It describes the applications, installation and configuration steps, operation examples, Web UI demo, and troubleshooting aspects of NETCONF. It also describes how to enable the NETCONF feature in OpenIL.

6.1 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 20. 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.

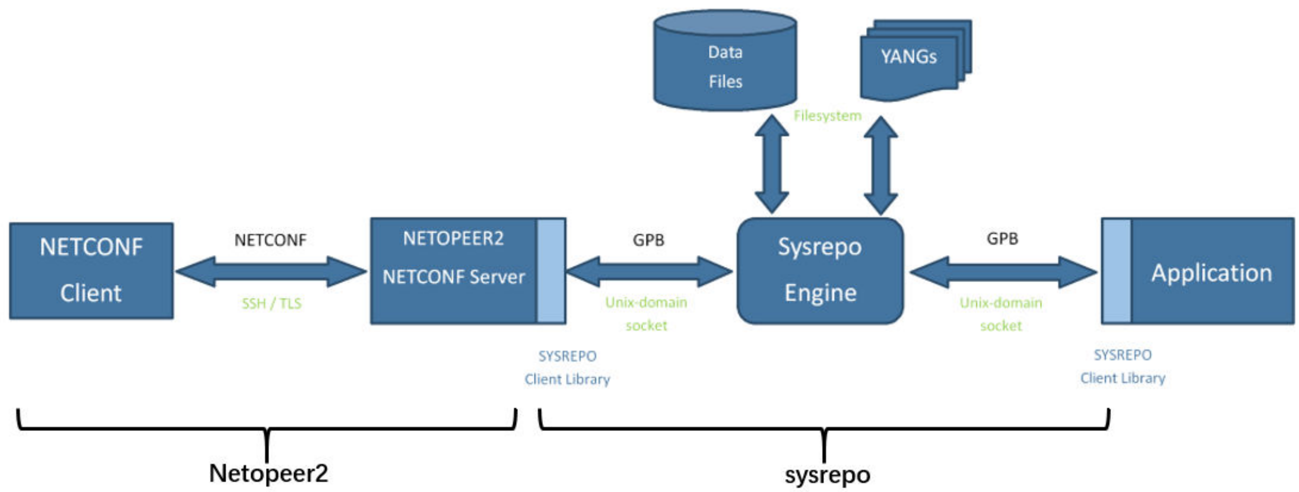
For information about YANG, see RFC 6020, YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF), and related RFCs.

6.2 Netopeer2

6.2.1 Overview

Netopeer2 is a set of tools implementing network configuration tools based on the NETCONF Protocol. This is the second generation of the toolset, originally available as the Netopeer project. It is based on the new generation of the NETCONF and YANG libraries - **libyang** and **libnetconf2**. The Netopeer2 server uses **sysrepo** as a NETCONF datastore implementation. In OpenIL-V1.7, version **v0.7-r2** was used. It allows developers to control their devices via NETCONF and operators to connect to their NETCONF-enabled devices.

Figure 19. High level architecture of Netopeer and sysrepo



6.2.2 Sysrepo

Sysrepo is an YANG-based configuration and operational state data store for Unix/Linux applications.

Applications can use sysrepo to store their configuration modeled by provided YANG model instead of using e.g. flat configuration files. In OpenIL-V1.7, version **v0.7.8** was used. Sysrepo will ensure data consistency of the data stored in the datastore and enforce data constraints defined by YANG model. Applications can currently use C language API of sysrepo Client Library to access the configuration in the datastore, but the support for other programming languages is planed for later too (since sysrepo uses Google Protocol Buffers as the interface between the datastore and client library, writing of a native client library for any programing language that supports GPB is possible).

For information about sysrepo, see:

http://www.sysrepo.org/static/doc/html/index.html

6.2.3 Netopeer2 server

Netopeer2 software is a collection of utilities and tools to support the main application, Netopeer2 server, which is a NETCONF server implementation. It uses libnetconf2 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 Netopeer2 server responds with correct replies.

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

- Netopeer2-keystored as a tool for the storage and process of keys.

- Netopeer2-server as the main service daemon integrating the SSH/TLS server.

6.2.4 Netopeer2 client

Netopeer2-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 Netopeer2 software bundle, but compiled and installed separately. It is a NETCONF client with a command line interface developed and primarily used for Netopeer2 server testing, but allowing all the standards and even some optional features of a full-fledged NETCONF client.

Netopeer2-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.

6.2.5 Workflow in application practice

In practical application, we use the YANG language to model the device and generate the YANG model. The model is then instantiated to generate configuration files in XML format. The device was then configured using this configuration file as input via netopeer.

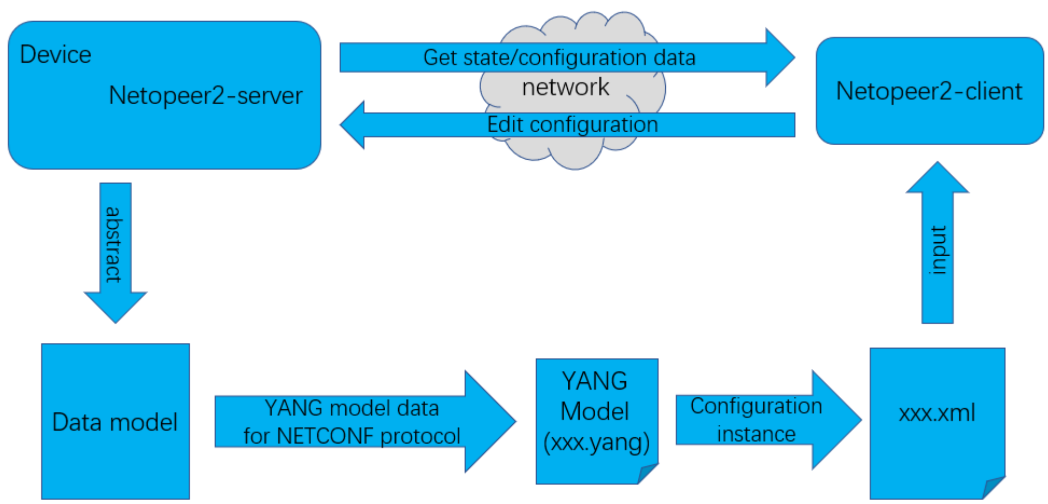


Figure 20. Workflow for netopeer

6.3 Installing Netopeer2-cli on Ubuntu18.04

Use the following steps for installing Netopeer2-cli onUbuntu18.04 operating systems.

1. Install the following packages:

```
$ sudo apt install -y git cmake build-essential bison autoconf dh-autoreconf flex
$ sudo apt install -y libavl-dev libprotobuf-c-dev protobuf-c-compiler zlib1g-dev
$ sudo apt install -y libgcrypt20-dev libssh-dev libev-dev libpcre3-dev
```

2. Install libyang:

```
$ git clone https://github.com/CESNET/libyang.git
$ cd libyang;
$ git checkout v1.0-r4 -b v1.0-r4
$ mkdir build; cd build
$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr ..
```

```
$ make
$ sudo make install
```

3. Install sysrepo(v0.7.8):

```
$ git clone https://github.com/sysrepo/sysrepo.git
$ cd sysrepo
$ git checkout v0.7.8 -b v0.7.8
$ mkdir build; cd build
$ cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX:PATH=/usr ..
$ make
$ sudo make install
```

4. Install libnetconf2:

```
$ git clone https://github.com/CESNET/libnetconf2.git
$ cd libnetconf2
$ git checkout v0.12-r2 -b v0.12-r2
$ mkdir build; cd build
$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr ..
$ make
$ sudo make install
```

5. Install protobuf:

```
$ git clone https://github.com/protocolbuffers/protobuf.git
$ cd protobuf
$ git submodule update --init --recursive
$ ./autogen.sh
$ ./configure
$ make
$ sudo make install
$ sudo ldconfig # refresh shared library cache.
```

6. Install Netopeer2-cli(v0.7-r2):

```
$ git clone https://github.com/CESNET/Netopeer2.git
$ cd Netopeer2
$ git checkout v0.7-r2 -b v0.7-r2
$ cd cli
$ cmake -DCMAKE_INSTALL_PREFIX:PATH=/usr .
$ make
$ sudo make install
```

6.4 Configuration

6.4.1 Enabling NETCONF feature in OpenIL

Build the image using the below command to enable the NETCONF feature:

```
make nxp_ls1028ardb-64b_defconfig
```

Users can find detailed configuration with the `make menuconfig` command, as shown below:

```
Target packages ->
  Netconf Supporting ->
      [ ] pyang
```

```
        [ ] libnetconf
        [ ] netopeer-server
        [ ] netopeer-client
        [ ] Netopeer2-cli
        -*- Netopeer2-keystored
        -*- Netopeer2-server
        [ ] yang-cfginterfaces
        [ ] yang-sja1105
        -*- yang-model
        [ ] transapi
        [*] sysrepo-tsn
```

sysrepo-tsn is daemon application to implement tsn configuration based on **sysrepo**. It was enabled in **nxp_ls1028ardb-64b_defconfig**. Presently, it only supports LS1028ARDB platform.

NOTE

- In OpenIL-v1.7-202001 version, only Qbv, Qbu, Qci, and stream identification in CB are supported.
- sysrepo-tsn is only verified in environment build by **nxp_ls1028ardb-64b_defconfig** configuration.

6.4.2 Netopeer2-server

The netopeer2-server is the NETCONF protocol server running as a system daemon. The netopeer2-server is based on sysrepo and libnetconf2 library.

- -U listen locally on a unix socket
- -d debug mode (do not daemonize and print verbose messages to stderr instead of syslog)
- -V: Show program version.
- -v level verbose output level(0 : errors, 1 : errors and warnings, 2 : errors, warnings and verbose messages).

6.4.3 Netopeer2-cli

The netopeer2-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. netopeer2-cli is limited to a single NETCONF connection at a time via a forward or a reverse (Call Home) connecting method.

6.4.3.1 Netopeer2 CLI commands

Following are the Netopeer2 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] [--ssh] [--host <hostname>] [--port <num>] [--login <username>]
   ```

   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.
   - **--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 or ip-address 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] --target running|startup|candidate|url:<url> (--source running|startup|
candidate|url:<url> | --src-config[=<file>])
    [--defaults report-all|report-all-tagged|trim|explicit]
```

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.
- **--target** datastore: Specifies the target datastore for the `copy-config` operation. For description of the datastore parameter, refer to Netopeer2 CLI datastore.
- **--source** datastore: Specifies the source datastore for the `copy-config` operation. For description of the datastore parameter, refer to Netopeer2 CLI datastore.

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

```
delete-config [--help] --target startup|url:<url>
```

Where

- **target** datastore: Specifies the target datastore for the `delete-config` operation.

7. **edit-config**

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

```
edit-config [--help] --target running|candidate (--config[=<file>] | --url <url>)
    [--defop merge|replace|none] [--test set|test-only|test-then-set] [--error stop|continue|
rollback]
```

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.

NOTE

ITo delete non-madatory items, **nc:operation="delete"** needs to be added into the end of start tag of the item to be deleted. At the same time, the namespace **xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0"** also needs to be added ioto start tag of the root node. Mandatory items can't be deleted individually, they can only be deleted with their parent node.

- **--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**

  — Target datastore to modify. For description of possible values, refer to Netopeer2 CLI datastore. 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] [--filter-subtree[=<file>] | --filter-xpath <XPath>] [--defaults report-all|report-all-tagged|trim|explicit] [--out <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.

9. **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] --source running|startup|candidate [--filter-subtree[=<file>] | --filter-xpath <XPath>]
    [--defaults report-all|report-all-tagged|trim|explicit] [--out <file>]
```

10. **--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.

11. **--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.
12. **--target**
    - Target datastore to retrieve. For description of possible values, refer to Netopeer2 CLI datastore. Note, that the url configuration datastore cannot be retrieved.
13. **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 running|startup|candidate
    ```

    Where the

    - **--target**: specifies the target datastore to lock. For description of possible values, refer to Netopeer2 CLI datastore. Note, that the url configuration datastore cannot be locked.
14. **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 running|startup|candidate
    ```

    where

    - **--target**: specifies the target datastore to unlock. For description of possible values, refer to Netopeer2 CLI datastore. Note, that the url configuration datastore cannot be unlocked.
15. **verb**
    - Enables/disables verbose messages.
16. **quit**
    - Quits the program.

6.4.3.2 Netopeer2 CLI datastore

Following are the netopeer2 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.

6.4.4 Sysrepod

Sysrepo deamon provides the functionality of the datastore on the system (executes the system-wide Sysrepo Engine) and should normally be automatically started by system startup. However, auto-start is not configured by cmake install operation and you need to configure it yourself, accroding to the guidelines of your system.

Usage: sysrepod [-h] [-v] [-d] [-l <level>]

Options:

- -h Prints usage help.
- -v Prints version.
- -d Debug mode - daemon will run in the foreground and print logs to stderr instead of syslog.
- -l <level> Sets verbosity level of logging:

  0 = all logging turned off

  1 = log only error messages

  2 = (default) log error and warning messages

  3 = log error, warning and informational messages

  4 = log everything, including development debug messages

6.4.5 Sysrepocfg

sysrepocfg is a command-line tool for editing, importing and exporting configuration stored in Sysrepo datastore. It allows the user to edit startup or running configuration of specified module in preferred text editor and to propagate the perfomed changes into the datastore transparently for all subscribed applications. Moreover, the user can export the current configuration into a file or get it printed to the standard output and, similarly, an already prepared configuration can be imported from a file or read from the standard input.

In the background, **sysrepocfg** uses Sysrepo client library for any data manipulation rather than directly accessing configuration data files, thus effectively inheriting all main features of Sysrepo, such as YANG-based data validation, full transaction and concurrency support, and, perhaps most importantly, subscribed applications are notified about the changes made using \fBsysrepocfg\fP and can immediately take the new configuration into account.

6.4.6 Sysrepoctl

The sysrepoctl provides fuctions to manage modules. It can be configured using the options and commands described below.

operation-operations

- **--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.
- **--install**: Installs specified schema into sysrepo (--yang or --yin must be specified).
- **--uninstall**: Uninstalls specified schema from sysrepo (--module must be specified).
- **--list**: Lists YANG modules installed in sysrepo (note that Conformance Installed implies also Implemented).
- **--change** : Changes specified module in sysrepo (--module must be specified).
- **--feature-enable**: Enables a feature within a module in sysrepo (feature name is the argument, --module must be specified).
- **--feature-disable**: Disables a feature within a module in sysrepo (feature name is the argument, --module must be specified).

Other-options

- **--yang** : Path to the file with schema in YANG format (--install operation).
- **--yin** : Path to the file with schema in YIN format (--install operation).
- **--module** : Name of the module to be operated on (--change, --feature-enable, --feature-disable operations, --uninstall - several modules can be delimited with ',').
- **--permissions** : Access permissions of the module's data in chmod format (--install, --change operations).

Examples

- Install a new module by specifying YANG file, ownership and access permissions:

  sysrepoctl --install --yang=/home/user/ietf-interfaces.yang --owner=admin:admin --permissions=644
- Change the ownership and permissions of an existing YANG module:

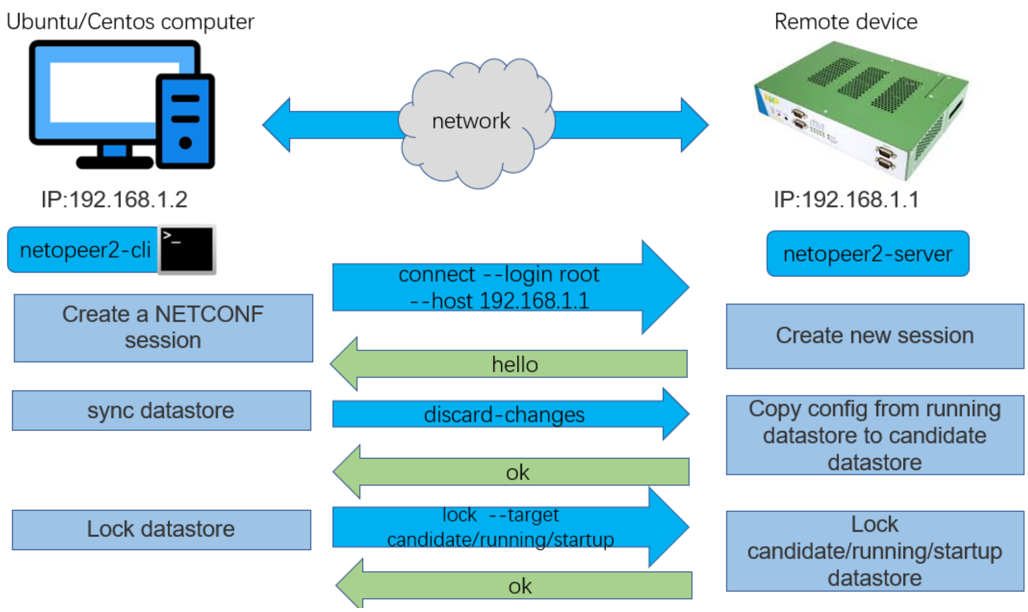
  sysrepoctl --change --module=ietf-interfaces --owner=admin:admin --permissions=644
- Enable a feature within a YANG module:

  sysrepoctl --feature-enable=if-mib --module=ietf-interfaces
- Uninstall 2 modules, second one is without revision:

  sysrepoctl --uninstall --module=mod-a,mod-b --revision=2035-05-05

6.4.7 Operation examples

The following figure describes the steps to configure device via netopeer2:



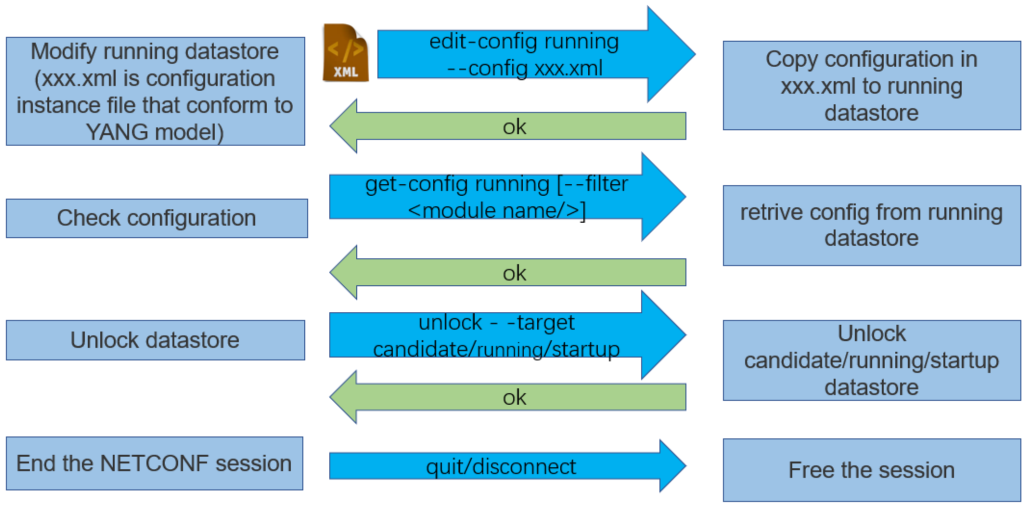


Figure 21. Steps to configure device via netopeer2

In sysrepo-tsn, there are some instance files to configure TSN features on LS1028ARDB board:

- Instance files for TSN configuration

Users can configure TSN functions of LS1028ARDB board using these instance files. Before starting, make sure that **sysrepod**, **sysrepo-plugind**, **sysrepo-tsn** and **netopeer2-server** are running on the board. Use the following steps to configure TSN feature on LS1028ARDB board.

1. Start netopeer2-cli on the computer with netopeer2-cli installed:

```
$ netopeer2-cli
```

2. Connect to netopeer2-server on LS1028ARDB board (use the IP on LS1028ARDB, here 10.193.20.53 is example):

```
> connect --login root --host 10.193.20.53
```

3. Get status data of server:

```
> get
```

4. Get configuration data in running datastore:

```
> get-config --source running
```

5. Cofigure QBV feature of LS1028ARDB with qbv-eno0-enable.xml

```
> edit-config --target running --config=qbv-eno0-enable.xml
```

6. Check configuration data of QBV:

```
> get-config --source running --filter-xpath /ietf-interfaces:interfaces/interface[name='eno0']/
ieee802-dot1q-sched:gate-parameters
```

7. Copy configuration data in **running** datastore to **startup** datastore:

```
> copy-config --source running --target startup
```

8. Disconnect with netopeer2-server:

```
> disconnect
```

> **NOTE**
>
> **Sysrepo-tsn** does not support LS1021ATSN board. Hence, if you want to configure LS1021ATSN via NETCONF toolset, please switch to *OpenIL-201908* version.

6.5 Web UI demo

The Web UI allows the remote control of the YANG model. This demo is already added to tsntool (https://github.com/openil/tsntool) in the folder `tsntool/demos/cnc/`. Follow the procedure mentioned below for this demo.

1. Install related libraries

   Suppose you are installing the demo on a Centos PC or Ubuntu PC as the WebServer. CNC demo requires python3 and related libraries:`pyang`, `libnetconf`, and `libssh`.

   For Ubuntu18.04

```
$ sudo apt install -y libtool python-argparse libtool-bin python-sphinx libffi-dev
$ sudo apt install -y libxslt1-dev libcurl4-openssl-dev xsltproc python-setuptools
$ sudo apt install -y zlib1g-dev libssl-dev python-libxml2 libaugeas-dev
$ sudo apt install -y libreadline-dev python-dev pkg-config libxml2-dev
$ sudo apt install -y cmake openssh-server
$ sudo apt install -y python3-sphinx python3-setuptools python3-libxml2
$ sudo apt install -y python3-pip  python3-dev python3-flask
$ sudo apt install -y libnss-mdns avahi-utils
```

   For Centos 7.2

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

2. Install pyang

```
$ git clone https://github.com/mbj4668/pyang.git
$ cd pyang
$ git checkout b92b17718de53758c4c8a05b6818ea66fc0cd4d8 -b fornetconf1
$ sudo python setup.py install
```

3. Install libssh

```
$ git clone https://git.libssh.org/projects/libssh.git
$ cd libssh
$ git checkout fe18ef279881b65434e3e44fc4743e4b1c7cb891  -b fornetconf1
$ mkdir build; cd build/
$ cmake ..
$ make
$ sudo make 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.

4. Install libnetconf

```
$ git clone https://github.com/CESNET/libnetconf.git
$ cd libnetconf
$ git checkout 8e934324e4b1e0ba6077b537e55636e1d7c85aed -b fornetconf1
$ autoreconf --force --install
$ ./configure
$ make
$ sudo make install
```

5. Get tsntool source code

```
git clone https://github.com/openil/tsntool.git
cd tsntool/demos/cnc/
```

6. Install python library

   In the below command segments,

   - `PATH-to-libnetconf` is the path to the libnetconf source code.
   - `PATH-to-tsntool` is the path to the tsntool source code.

```
$ cd PATH-to-libnetconf/libnetconf
```

   The `libnetconf` needs to add two patches based on the below commit point to fix the demo python support.

   Ensure that the commit id is `313fdadd15427f7287801b92fe81ff84c08dd970`.

```
$ git checkout 313fdadd15427f7287801b92fe81ff84c08dd970 -b cnc-server
$ cp PATH-to-tsntool/demos/cnc/*patch .
$ git am 0001-lnctool-to-make-install-transapi-yang-model-proper.patch
$ git am 0002-automatic-python3-authorizing-with-root-password-non.patch
$ cd PATH-to-libnetconf/libnetconf/python
$ python3 setup.py build; sudo python3 setup.py install
```

NOTE

If rebuilding python lib, you need to remove the `build` folder by command `rm build -rf` before rebuilding. On the OpenIL board, `avahi-daemon` and netopeer server are required. Remember to also add the netopeer2-server run at boards.

7. Setup `avahi daemon` and disable the ipv6:

   For this, edit `/etc/avahi/avahi-daemon.conf`

```
use-ipv6=no
publish-a-on-ipv6=no
```

```
sudo systemctl start avahi-daemon.service
```

8. Packages required by OpenIL Board

```
On the OpenIL board, avahi-daemon, and netopeer server are required:
BR2_PACKAGE_AVAHI=y
```

```
BR2_PACKAGE_AVAHI_AUTOIPD=y
BR2_PACKAGE_AVAHI_DAEMON=y
BR2_PACKAGE_AVAHI_LIBDNSSD_COMPATIBILITY=y
BR2_PACKAGE_NSS_MDNS=y
BR2_PACKAGE_NETOPEER2_SERVER=y
```

Openil update the netopeer server to version2. Remember to make the netopeer2-server run at boards.

9. Start the web server
   - Input the command below at shell into the folder `/tsntool/demos/cnc/`:

     ```
     sudo python3 cnc.py
     ```

   - Then, input the IP of WebServer with the port 8180 at browser. For example:

     ```
     http://10.193.20.147:8180
     ```

   - It is recommended to tracking the boards by tsntool to checking the real configuration for comparison.
   - It is also recommended to tracking if the netopeer2-server is running ata board or not.

NOTE

Limitations of Web UI are:

- Setup server on a Centos PC or Ubuntu PC could be more compatible.
- Supports Qbv, Qbu and Qci in current version.
- For Qci setting, Stream-gate entry should be set ahead of setting the Stream-filter as sysrepo required. Or else, you will got failure for setting Stream-filter without a stream gate id link to.
- The boards and the web server PC are required to be in same IP domain since the bridge may block the probe frames.

6.6 Troubleshooting

1. Connect fails at client side:

```
nc ERROR: Remote host key changed, the connection will be terminated!
nc ERROR: Checking the host key failed.
cmd_connect: Connecting to the 10.193.20.4:830 as user "root" failed.
```

Fixing:

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

- You need to get host list with command `knownhosts` first.
- Then remove related item. For example **knownhosts --del 19**.

2. Request could not be completed because the relevant data model content does not exist.

```
type:     application
tag:      data-missing
severity: error
path:     /ietf-interfaces:interfaces/interface[name='eno0']/ieee802-dot1q-sched:gate-parameters/
admin-gate-states
message:  Request could not be completed because the relevant data model content does not exist.
```

Fixing:

The reason is that the configuration data in xpath does not exist in the datastore. Such as deleting a node that does not exist.

When encountering such an error, user can get configuration data in the board with **get-config** command, and check whether the operation type(add/delete/modify) of the node in the path is reasonable or not,.

Chapter 7
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

7.1 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.

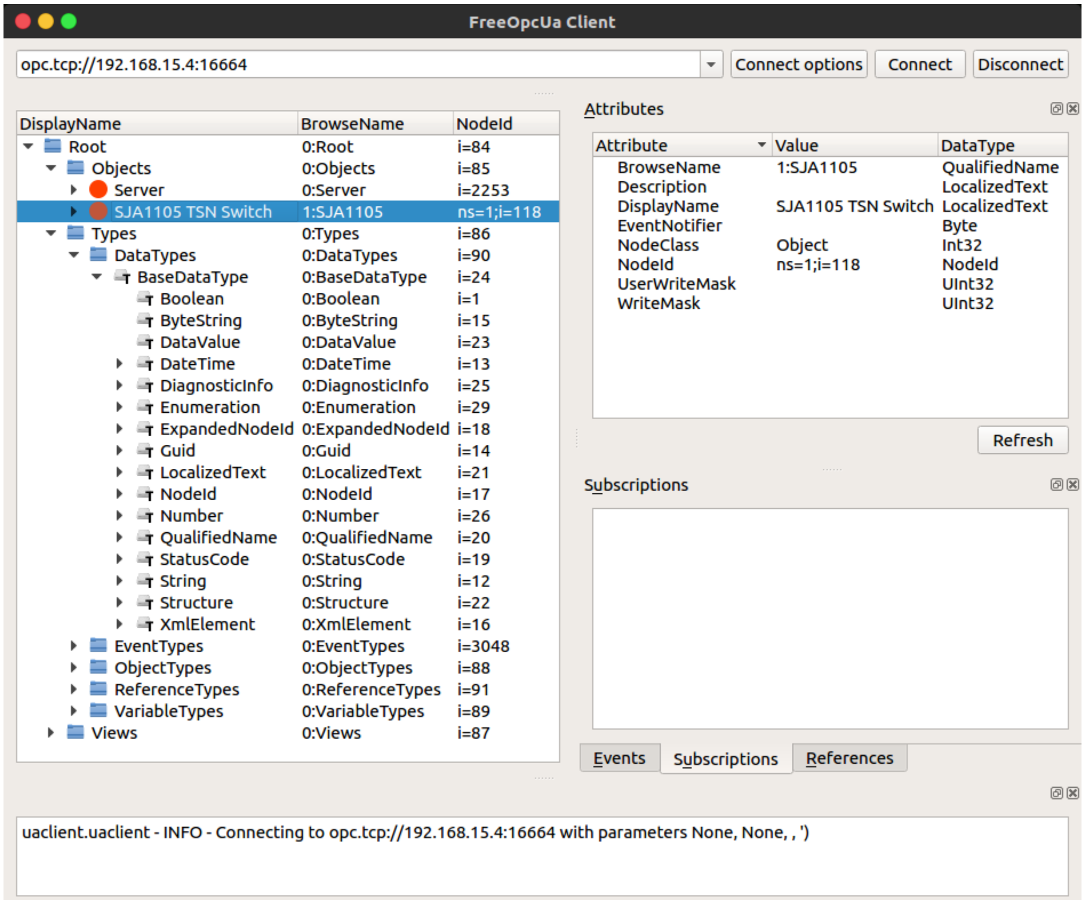
7.2 The node model

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 22. 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`.

7.3 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*.

7.4 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*.

7.5 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.

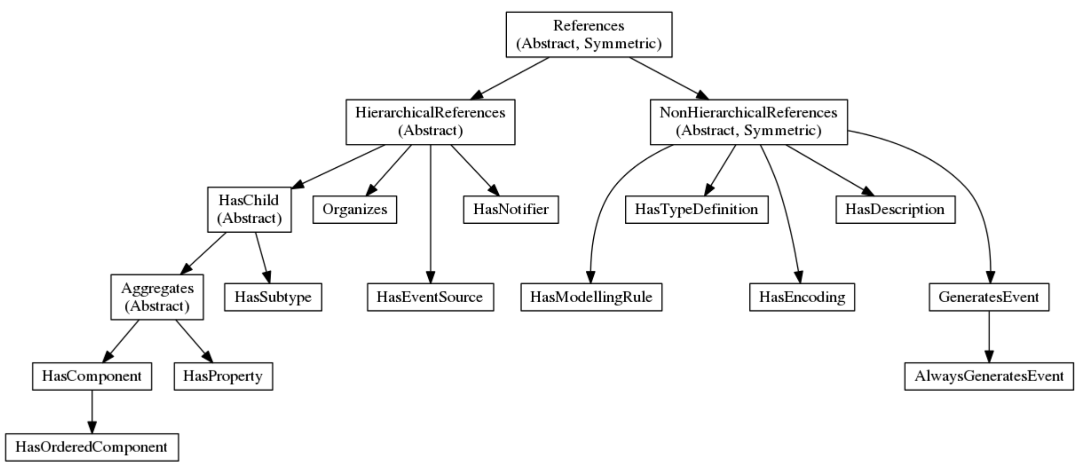


Figure 23. Hierarchy of the standard ReferenceTypes, defined in Part 3 of the OPC UA specification (Image taken from www.open62541.org)

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.

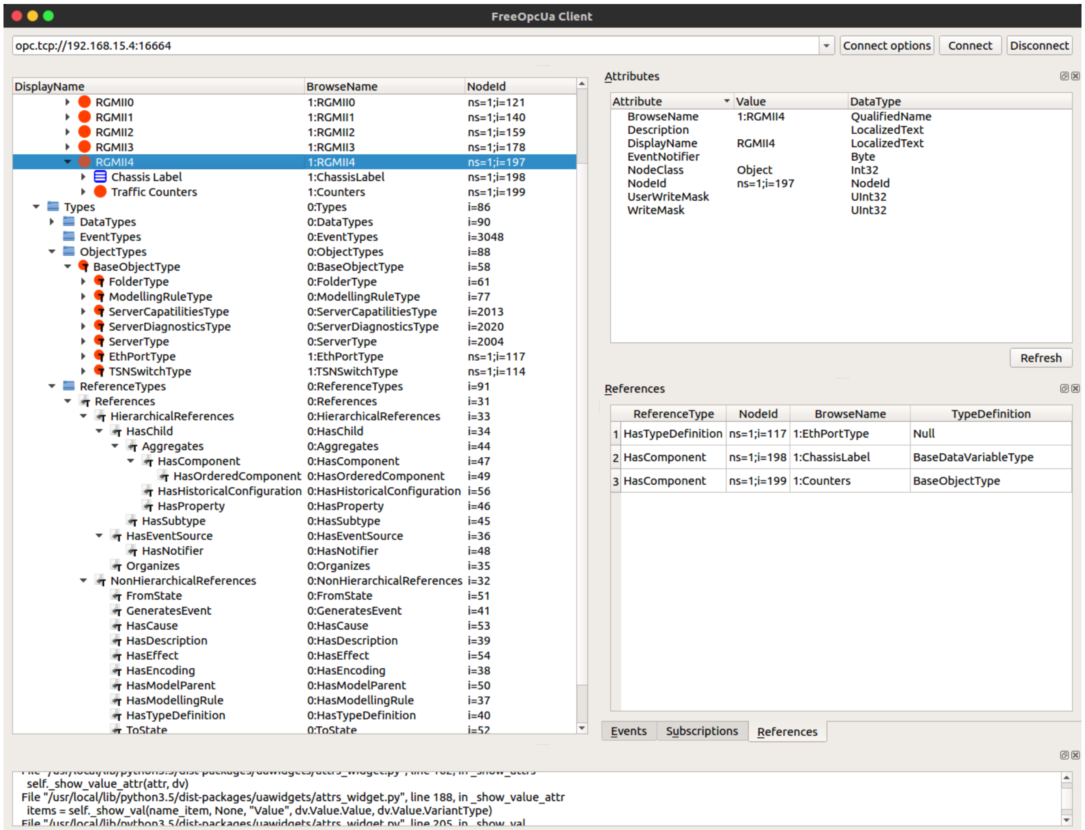


Figure 24. The 'Attributes' and 'References' views of the FreeOpcUa Client populated with details of the RGMII4 node

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`.

7.6 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

7.7 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.

7.8 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:

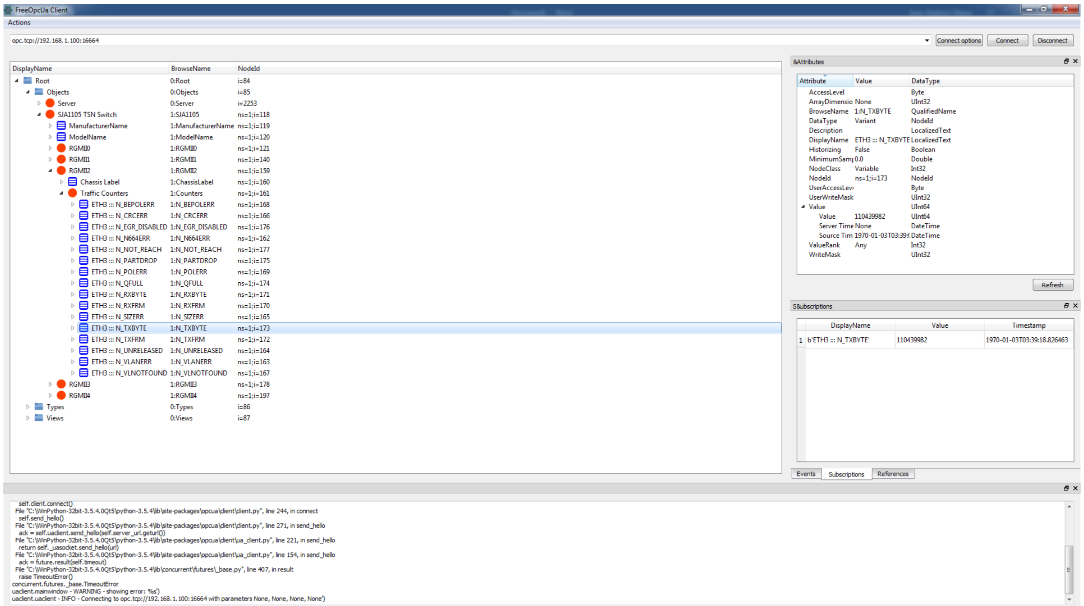


Figure 25. Subscribing to data changes of the ETH3 ::: N_TXBYTE node of the OPC-SJA1105 server

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.

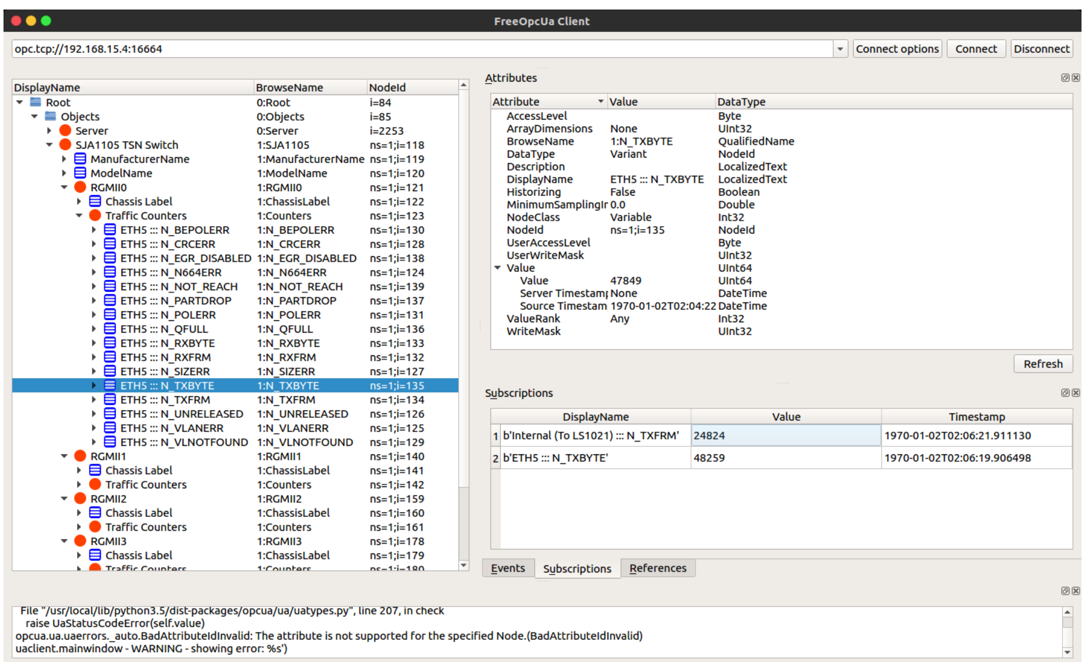


Figure 26. Data change notification

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 8
TSN

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.

This chapter describes the process and use cases for implementing TSN features on the LS1021ATSN and the LS1028ARDB boards.

8.1 Using TSN features on LS1028ARDB

The **tsntool** is an application configuration tool to configure the TSN capability on LS1028ARDB. The files **/usr/bin/tsntool** and **/usr/lib/libtsn.so** are located in the rootfs. Run **tsntool** to start the setting shell.

8.1.1 Tsntool User Manual

Tsntool is a tool to set the TSN capability of the Ethernet ports of TSN Endpoint and TSN switch. This document describes how to use tsntool for NXP's LS1028ARDB hardware platform.

NOTE

Currently the Tsntool supports only the LS1028ARDB platform. Other hardware platforms might be supported in future.

8.1.1.1 Getting the source code

Github of the tsntool code is:

https://github.com/openil/tsntool.git

8.1.1.2 Tsn tool commands

The following table lists the TSN tool commands and their description.

Table 21. TSN tool commands and their description

Command	Description
help	Lists commands support
version	Shows software version
verbose	Debugs on/off for tsntool
quit	Quits prompt mode
qbvset	Sets time gate scheduling config for `<ifname>`
qbvget	Gets time scheduling entries for `<ifname>`
cbstreamidset	Sets stream identification table

Table continues on the next page...

Table 21. TSN tool commands and their description (continued)

Command	Description
cbstreamidget	Gets stream identfication table and counters
qcisfiset	Sets stream filter instance
qcisfiget	Gets stream filter instance
qcisgiset	Sets stream gate instance
qcisgiget	Gets stream gate instance
qcisficounterget	Gets stream filter counters
qcifmiset	Sets flow metering instance
qcifmiget	Gets flow metering instance
cbsset	Sets TCs credit-based shaper configure
cbsget	Gets TCs credit-based shaper status
qbuset	Sets one 8-bits vector showing the preemptable traffic class
qbugetstatus	Not supported
tsdset	Not supported
tsdget	Not supported
ctset	Sets cut through queue status (specific for ls1028 switch)
cbgen	Sets sequence generate configure (specific for ls1028 switch)
cbrec	Sets sequence recover configure (specific for ls1028 switch)
dscpset	Sets queues map to DSCP of Qos tag (specific for ls1028 switch)
sendpkt	Not supported
regtool	Register read/write of bar0 of PFs (specific for ls1028 enetc)

ptptool	ptptool get/set ptp timestamp. Useful commands:
	`#get ptp0 clock time`
	`ptptool -g`
	
	`#get ptp1 clock time`
	`ptptool -g -d /dev/ptp1`

dscpset	Set queues map to DSCP of QoS tag (specific for ls1028 switch)

Table continues on the next page...

Table 21. TSN tool commands and their description (continued)

Command	Description
qcicapget	Gets qci instance's max capability
tsncapget	Gets device's tsn capability

8.1.1.3 Tsntool commands and parameters

This section lists the tsntool commands along with the parameters and arguments, with which they can be used.

Table 22. qbvset

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--entryfile <filename>	A file script to input gatelist format. It has the following arguments: `#'NUMBER' 'GATE_VALUE' 'TIME_LONG'` • NUMBER: # 't' or 'T' head. Plus entry number. Duplicate entry number will result in an error. • GATE_VALUE: # format: xxxxxxxxb . # The MSB corresponds to traffic class 7. The LSB corresponds to traffic class 0. # A bit value of 0 indicates closed, whereas, a bit value of 1 indicates open. • TIME_LONG: # nanoseconds. Do not input 0 time long. t0 11101111b 10000 t1 11011111b 10000 NOTE `Entryfile` parameter must be set. If not set, there will be a vi text editor prompt, "`require to input the gate list`".
--basetime <value>	AdminBaseTime A 64-bit hex value means nano second until now. OR a value input format as: `Seconds.decimalSecond` Example: `115.000125`means 115seconds and 125us.
--cycletime <value>	AdminCycleTime
--cycleextend <value>	AdminCycleTimeExtension
--enable | --disable	• enable: enables the `qbv` for this port • disable: disables the `qbv` for this port Default is set to enable, if no enable or disable input
--maxsdu <value>	queueMaxSDU
--initgate <value>	AdminGateStates
--configchange	ConfigChange. Default set to 1.
--configchangetime <value>	ConfigChangeTime

Table 23. qbvget

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`

Table 24. cbstreamidset

Parameter <argument>	Description
--enable | --disable	• enable: Enables the entry for this index. • disable: Disables the entry for this index. Default is set to `enable` if no enable or disable input
--index <value>	Index entry number in this controller. Mandatory parameter. This value corresponds to `tsnStreamIdHandle` on switch configuration.
--device <string>	An interface such as `eno0/swp0`
--streamhandle <value>	tsnStreamIdHandle
--infacoutport <value>	tsnStreamIdInFacOutputPortList
--outfacoutport <value>	tsnStreamIdOutFacOutputPortList
--infacinport <value>	tsnStreamIdInFacInputPortList
--outfacinport <value>	tsnStreamIdOutFacInputPortList
--nullstreamid | --sourcemacvid | --destmacvid | --ipstreamid	tsnStreamIdIdentificationType: • -nullstreamid:Null Stream identification • -sourcemacvid: Source MAC and VLAN Stream identification • -destmacvid: not supported • -ipstreamid: not supported
--nulldmac <value>	tsnCpeNullDownDestMac
--nulltagged <value>	tsnCpeNullDownTagged
--nullvid <value>	tsnCpeNullDownVlan
--sourcemac <value>	tsnCpeSmacVlanDownSrcMac
--sourcetagged <value>	tsnCpeSmacVlanDownTagged
--sourcevid <value>	tsnCpeSmacVlanDownVlan

Table 25. cbstreamidget

P arameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--index <value>	Index entry number in this controler. Mandatory to have.

Table 26. qcisfiset

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--enable | --disable	• enable: enable the entry for this index • disable: disable the entry for this index • default to set `enable` if no enable or disable input
--maxsdu <value>	Maximum SDU size.
--flowmeterid <value>	Flow meter instance identifier index number.
--index <value>	StreamFilterInstance. index entry number in this controler. This value corresponds to `tsnStreamIdHandle` of `cbstreamidset` command on switch configuration.
--streamhandle <value>	StreamHandleSpec This value corresponds to `tsnStreamIdHandle` of `cbstreamidset` command.
--priority <value>	PrioritySpec
--gateid <value>	StreamGateInstanceID
--oversizeenable	StreamBlockedDueToOversizeFrameEnable
--oversize	StreamBlockedDueToOversizeFrame

Table 27. qcisfiget

parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--index <value>	Index entry number in this controller. Mandatory to have.

Table 28. qcisgiset

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--index <value>	Index entry number in this controller. Mandatory to have.
--enable | --disable	• enable: enable the entry for this index. PSFPGateEnabled • disable: disable the entry for this index • default to set enable if no enable or disable input
--configchange	configchange
--enblkinvrx	PSFPGateClosedDueToInvalidRxEnable
--blkinvrx	PSFPGateClosedDueToInvalidRx
--initgate	PSFPAdminGateStates
--initipv	AdminIPV
--cycletime	Default not set. Get by gatelistfile.
--cycletimeext	PSFPAdminCycleTimeExtension
--basetime	PSFPAdminBaseTime A 64-bit hex value means nano second until now. OR a value input format as: `Seconds.decimalSecond` Example: `115.000125`means 115seconds and 125us.
--gatelistfile	PSFPAdminControlList. A file input the gate list: 'NUMBER' 'GATE_VALUE' 'IPV' 'TIME_LONG' 'OCTET_MAX' • NUMBER: # 't' or 'T' head. Plus entry number. Duplicate entry number will result in an error. • GATE_VALUE: format: xb: The MSB corresponds to traffic class 7. The LSB corresponds to traffic class 0. A bit value of 0 indicates closed, A bit value of 1 indicates open. • IPV: # 0~7 • TIME_LONG: in nanoseconds. Do not input time long as 0. • OCTET_MAX: The maximum number of octets that are permitted to pass the gate. If zero, there is no maximum. t0 1b -1 50000 10

Table 29. qcisgiget

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--index <value>	Index entry number in this controller. Mandatory to have.

Table 30. qcifmiset

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--index <value>	Index entry number in this controller. Mandatory to have.
--disable	If not set disable, then to be set enable.
--cir <value>	cir. kbit/s.
--cbs <value>	cbs. octets.
--eir <value>	eir.kbit/s.
--ebs <value>	ebs.octets.
--cf	cf. couple flag.
--cm	cm. color mode.
--dropyellow	drop yellow.
--markred_enable	mark red enable.
--markred	mark red.

Table 31. qcifmiget parameter

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--index <value>	Index entry number in this controller. Mandatory to have.

Table 32. qbuset parameter

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--preemptable <value>	8-bit hex value. Example: 0xfe The MS bit corresponds to traffic class 7. The LS bit to traffic class 0. A bit value of 0 indicates express. A bit value of 1 indicates preemptable.

Table 33. cbsset command

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`

Table continues on the next page...

Table 33. cbsset command (continued)

Parameter <argument>	Description
--tc <value>	Traffic class number.
--percentage <value>	Set percentage of tc limitation.
--all <tc-percent:tc-percent...>	Not supported.

Table 34. cbsget

Parameter <argument>	Description
--device <ifname>	An interface such as `eno0/swp0`
--tc <value>	Traffic class number.

Table 35. regtool

Parameter <argument>	Description
Usage: regtool { pf number } { offset } [data]	pf number: pf number for the pci resource to act on
	offset: offset into pci memory region to act upon
	data: data to be written

Table 36. ctset

Parameter <argument>	Description
--device <ifname>	An interface such as `swp0`
--queue_stat <value>	Specifies which priority queues have to be processed in cut-through mode of operation. Bit 0 corresponds to priority 0, Bit 1 corresponds to priority 1 so-on.

Table 37. cbgen

Parameter <argument>	Description
--device <ifname>	An interface such as `swp0`
--index <value>	Index entry number in this controller. Mandatory to have. This value corresponds to `tsnStreamIdHandle` of `cbstreamidset` command.
--iport_mask <value>	INPUT_PORT_MASK: If the packet is from input port belonging to this port mask, then it's a known stream and Sequence generation parameters can be applied
--split_mask <value>	SPLIT_MASK: Port mask used to add redundant paths (or ports). If split is enabled (STREAM_SPLIT) for a stream. This is OR'ed with the final port mask determined by the forwarding engine.

Table continues on the next page...

Table 37. cbgen (continued)

Parameter <argument>	Description
--seq_len <value>	SEQ_SPACE_LOG2: Minimum value is 1 and maximum value is 28. `tsnSeqGenSpace = 2**SEQ_SPACE_LOG2` For example, if this value is 12, then valid sequence numbers are from 0x0 to 0xFFF.
--seq_num <value>	GEN_REC_SEQ_NUM: The sequence number to be used for outgoing packet passed to `SEQ_GEN` function. Note: Only lower 16-bits are sent in RED_TAG.

Table 38. cbrec

Parameter <argument>	Description
--device <ifname>	An interface such as `swp0`
--index <value>	Index entry number in this controller. Mandatory to have. This value corresponds to `tsnStreamIdHandle` of `cbstreamidset` command.
--seq_len <value>	SEQ_SPACE_LOG2:Min value is 1 and maximum value is 28. `tsnSeqRecSeqSpace = 2**SEQ_REC_SPACE_LOG2` For example, if this value is 12, then valid sequence numbers are from 0x0 to 0xFFF.
--his_len <value>	SEQ_HISTORY_LEN: Refer to SEQ_HISTORY, Min 1 and Max 32.
--rtag_pop_en	REDTAG_POP: If True, then the redundancy tag is popped by rewriter.

Table 39. dscpset

Parameter <argument>	Description
--device <ifname>	An interface such as `swp0`
--disable	Disable DSCP to traffic class for frames.
--index	DSCP value
--cos	Priority number of queue which is mapped to
--dpl	Drop level which is mapped to

Table 40. qcicapget

Parameter <argument>	Description
--device <ifname>	An interface such as swp0

Table 41. tsncapget

Parameter <argument>	Description
--device <ifname>	An interface such as swp0

8.1.1.4 Input tips

While providing the command input, you can use the following shortcut keys to make the input faster:

- When you input a command, use the **TAB** key to help list the related commands.

  For example:

  ```
  tsntool> qbv
  ```

  Then press **TAB** key, to get all related `qbv*` start commands.

  If there is only one choice, it is filled as the whole command automatically.

- When you input parameters, if you don't remember the parameter name. You can just input "--" then press **TAB** key. It displays all the parameters.

  If you input half the parameter's name, pressing the **TAB** key lists all the related names.

- History: press the up arrow "↑" . You will get the command history and can re-use the command.

8.1.1.5 Non-interactive mode

Tsntool also supports non-interactive mode.

For example:

In the interactive mode:

```
tsntool> qbuset --device eno0 --preemptable 0xfe
```

In non-interactive mode:

```
tsntool qbuset --device eno0 --preemptable 0xfe
```

8.1.2 Kernel configuration

Before compiling the Linux kernel, we need to configure it. In the kernel, select the configuration settings displayed below:

```
 Symbol: TSN [=y]
    | Type : boolean
    | Prompt: 802.1 Time-Sensitive Networking support
    | Location:
    | -> Networking support (NET [=y])
    | -> Networking options
    | Depends on: NET [=y] && VLAN_8021Q [=y] && PTP_1588_CLOCK [=y]
|
  Symbol: ENETC_TSN [=y]
    | Type : boolean
    | Prompt: TSN Support for NXP ENETC driver
    | Location:
    | -> Device Drivers
    | -> Network device support (NETDEVICES [=y])
    | -> Ethernet driver support (ETHERNET [=y])
```

```
    | -> Freescale devices (NET_VENDOR_FREESCALE [=y])
    | Defined at drivers/net/ethernet/freescale/enetc/Kconfig:41
    | Depends on: NETDEVICES [=y] && ETHERNET [=y] && NET_VENDOR_FREESCALE [=y] && FSL_ENETC [=m] &&
TSN [=y]
|
  Symbol: FSL_ENETC_PTP_CLOCK [=y]
    | Type : tristate
    | Prompt: ENETC PTP clock driver
    | Location:
    | -> Device Drivers
    | -> Network device support (NETDEVICES [=y])
    | -> Ethernet driver support (ETHERNET [=y])
    | -> Freescale devices (NET_VENDOR_FREESCALE [=y])
|
  Symbol: FSL_ENETC_HW_TIMESTAMPING [=y]
    | Type : boolean
    | Prompt: ENETC hardware timestamping support
    | Location:
    | -> Device Drivers
    | -> Network device support (NETDEVICES [=y])
    | -> Ethernet driver support (ETHERNET [=y])
    | -> Freescale devices (NET_VENDOR_FREESCALE [=y])
|
   Symbol: MSCC_FELIX_SWITCH_TSN [=y]
    | Type : tristate
    | Prompt: TSN on FELIX switch driver
    | Location:
    | -> Device Drivers
    |   -> Network device support (NETDEVICES [=y])
    |    -> Ethernet driver support (ETHERNET [=y])
    |     -> Microsemi devices (NET_VENDOR_MICROSEMI [=y])
    |      -> Ocelot switch driver (MSCC_OCELOT_SWITCH [=y])
    |      -> FELIX switch driver (MSCC_FELIX_SWITCH [=y])
    | Defined at drivers/net/ethernet/mscc/Kconfig:38

Symbol: NET_PKTGEN [=y]
    | Type  : tristate
    | Prompt: Packet Generator (USE WITH CAUTION)
    | Location:
    |     -> Networking support (NET [=y])
    |     -> Networking options
    |     -> Network testing
    |  Defined at net/Kconfig:325
    |  Depends on: NET [=y] && INET [=y] && PROC_FS [=y]

Symbol: MSCC_FELIX_SWITCH_PTP_CLOCK [=y]
     | Type  : boolean
     | Prompt: FELIX switch PTP clock support
     | Location:
     | -> Device Drivers
     |    -> Network device support (NETDEVICES [=y])
     |        -> Ethernet driver support (ETHERNET [=y])
     |             -> Microsemi devices (NET_VENDOR_MICROSEMI [=y])
     |                   -> Ocelot switch driver (MSCC_OCELOT_SWITCH [=y])
     |                       -> FELIX switch driver (MSCC_FELIX_SWITCH [=y])
     | Defined at drivers/net/ethernet/mscc/Kconfig:38
     | Depends on: NETDEVICES [=y] && ETHERNET [=y] && NET_VENDOR_MICROSEMI
     | Selects: PTP_1588_CLOCK [=y]
```

8.1.3 Basic TSN configuration examples on ENETC

The tsntool is an application configuration tool to configure the TSN capability. You can find the file, `/usr/bin/tsntool` and `/usr/lib/libtsn.so` in the rootfs. Run `tsntool` to start the setting shell. The following sections describe the TSN configuration examples on the ENETC ethernet driver interfaces.

Before testing the ENETC TSN test cases, you need to enable `mqprio` by using the command:

```
tc qdisc add dev eno0 root handle 1: mqprio num_tc 8 map 0 1 2 3 4 5 6 7 hw 1
```

8.1.3.1 Linuxptp test

To test 1588 synchronization on ENETC interfaces, use the following procedure:

1. Connect ENETC interfaces on two boards in a back-to-back manner. (For example, eno0 to eno0.)

   The linux booting log is as follows:

   ```
   …
   pps pps0: new PPS source ptp0
   …
   ```

2. Check PTP clock and timestamping capability:

   ```
   # ethtool -T eno0
   Time stamping parameters for eno0:
   Capabilities:
       hardware-transmit          (SOF_TIMESTAMPING_TX_HARDWARE)
       hardware-receive           (SOF_TIMESTAMPING_RX_HARDWARE)
       hardware-raw-clock         (SOF_TIMESTAMPING_RAW_HARDWARE)
   PTP Hardware Clock: 0
   Hardware Transmit Timestamp Modes:
        off          (HWTSTAMP_TX_OFF)
        on           (HWTSTAMP_TX_ON)Hardware Receive Filter Modes:
        none         (HWTSTAMP_FILTER_NONE)
        all          (HWTSTAMP_FILTER_ALL)
   ```

3. Configure the IP address and run `ptp4l` on two boards:

   ```
   # ifconfig eno0 <ip_addr>
   # ptp4l -i eno0 -p /dev/ptp0 -m
   ```

4. After running, one board would be automatically selected as the master, and the slave board would print synchronization messages.

5. For 802.1AS testing, just use the configuration file `gPTP.cfg` in linuxptp source. Run the below command on the boards, instead:

   ```
   # ptp4l -i eno0 -p /dev/ptp0 -f gPTP.cfg -m
   ```

8.1.3.2 Qbv test

This test includes the Basic gates closing test, Basetime test, and the Qbv performance test. These are described in the following sections.

8.1.3.2.1 Basic gates closing

The commands below describe the steps for closing the basic gates:

```
cat > qbv0.txt << EOF
t0      00000000b               20000
EOF
```

```
#Explanation:
# 'NUMBER'        :     t0
# 'GATE_VALUE'    :     00000000b
# 'TIME_LONG'     :     20000 ns
```

```
cp libtsn.so /lib
./tsntool
tsntool> verbose
tsntool> qbvset --device eno0 --entryfile ./qbv0.txt

ethtool -S eno0
ping 192.168.0.2 -c 1   #Should not pass any frame since gates are all off.
```

8.1.3.2.2 Basetime test

Base on case 1 `qbv1.txt` gate list.

```
#create 1s gate
cat > qbv1.txt << EOF
t0  11111111b      10000
t1  00000000b      99990000
EOF

tsntool> regtool 0 0x18
tsntool> regtool 0 0x1c

#read the current time
tsntool> ptptool -g

#add some seconds, for example, you get 200.666 time clock, then set 260.666 as result

tsntool> qbvset --device eno0 --entryfile qbv1.txt --basetime 260.666
tsntool> qbvget --device eno0 #You can check configchange time
tsntool> regtool 0 0x11a10 #Check pending status, 0x1 means time gate is working

#Waiting to change state, ping remote computer
ping 192.168.0.2 -A -s 1000

#The reply time will be about 100 ms
```

Since 10000 ns is the maximum limit for package size 1250 B.

```
ping 192.168.0.2 -c 1 -s 1300 #frame should not pass
```

8.1.3.2.3 Qbv performance test

Use the setup described in the figure below for testing ENETC port0 (MAC0).

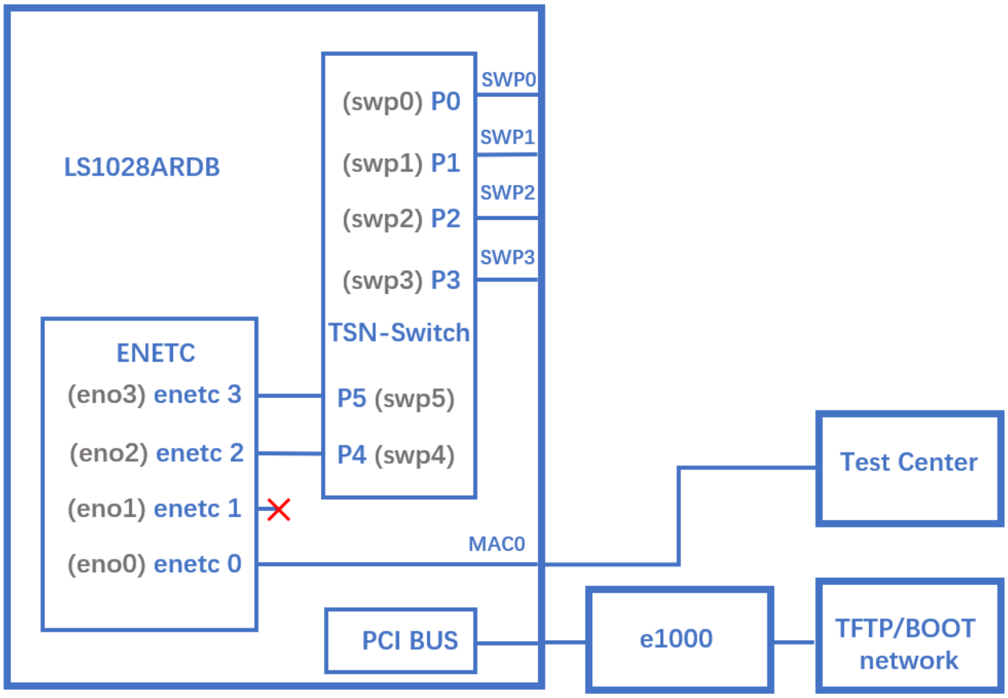


Figure 27. Setup for testing ENETC port0

```
cat > qbv5.txt << EOF
t0  11111111b      1000000
t1  00000000b      1000000
EOF

qbvset --device eno0 --entryfile qbv5.txt
./pktgen/pktgen_twoqueue.sh -i eno0 -q 3 -n 0

#The stream would get about half line rate
```

8.1.3.3 Qci test cases

Use the following as the background setting:

- Set eno0 MAC address

  ```
  ip link set eno0 address 10:00:80:00:00:00
  ```

  TestCenter MAC address 99:aa:bb:cc:dd:ee as an example.

- Use the figure below as the hardware setup.

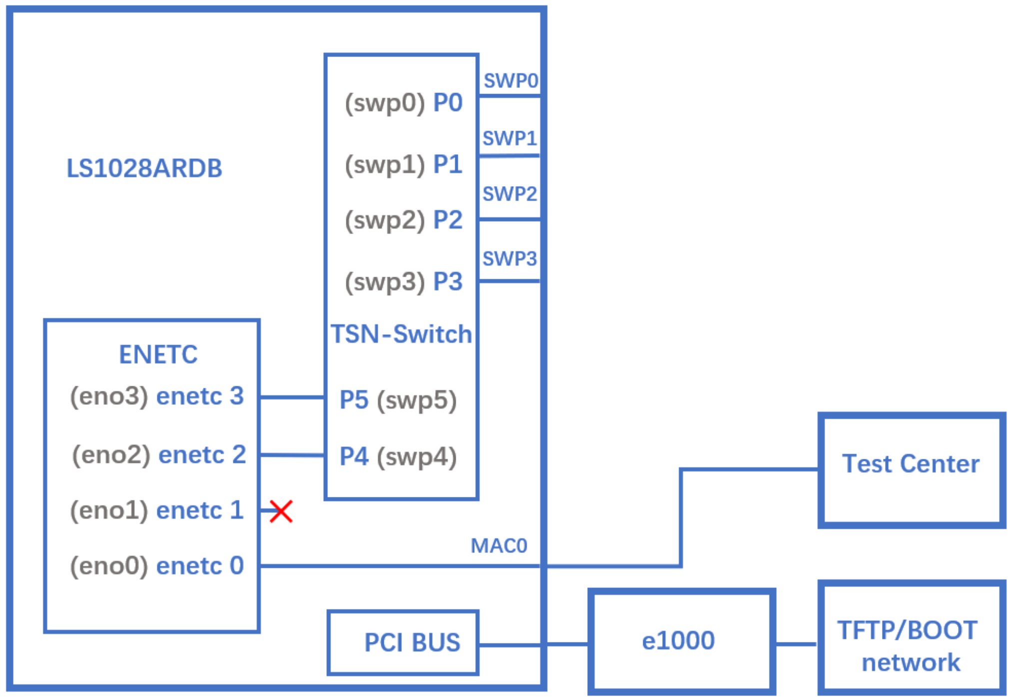


Figure 28. Qci test case setup

8.1.3.3.1 Test SFI No Streamhandle

To test no streamhandle for a stream filter, set a close gate stream id 2. Then no stream identifies the package check and other streams would pass the gate, as shown in the following example:

```
tsntool> qcisfiset --device eno0 --index 2 --gateid 2
```

- Streams no streamhandle should pass this filter.

```
tsntool> qcisfiget --device eno0 --index 2
```

- Send a frame from the Test center.

```
tsntool> qcisfiget --device eno0 --index 2
```

- Set Stream Gate entry 2

```
tsntool> qcisgiset --device eno0 --index 2 --initgate 1
```

- Send a frame from the Test center.

```
tsntool> qcisfiget --device eno0 --index 2
```

- Set Stream Gate entry 2, gate closes permanently.

```
tsntool> qcisgiset --device eno0 --index 2 --initgate 0
```

- Send a frame from the Test center.

```
tsntool> qcisfiget --device eno0 --index 2

#The result should look like below:
 match  pass  gate_drop  sdu_pass  sdu_drop red
   1    0   1     1    0    0
```

8.1.3.3.2 Testing null stream identify entry

Use the following steps:

1. Set main stream by close gate.
2. Set Stream identify Null stream identify entry 1.

```
tsntool> cbstreamidset --device eno0 --index 1 --nullstreamid --nulldmac
0x000000800010 --nulltagged 3 --nullvid 10 --streamhandle 100
```

3. Get SID index 1.

```
tsntool> cbstreamidget --device eno0 --index 1
```

4. Set Stream filer entry 1.

```
tsntool> qcisfiset --device eno0 --streamhandle 100 --index 1 --gateid 1
```

5. Set Stream Gate entry 1.

```
tsntool> qcisgiset --device eno0 --index 1 --initgate 0
```

6. Send one frame from the Test center.

```
tsntool> qcisfiget --device eno0 --index 1
```

7. The result should look like the output below:

```
match pass gate_drop sdu_pass sdu_drop red
1 0 1 1 0 0
```

8.1.3.3.3 Testing source stream identify entry

Use the following steps for this test:

1. Keep Stream Filter entry 1 and Stream gate entry 1.
2. Add stream2 in test center: `SMAC is 66:55:44:33:22:11 DMAC:20:00:80:00:00:00`
3. Set Stream identify Source stream identify entry 3

```
tsntool> cbstreamidset --device eno0 --index 3 --sourcemacvid --sourcemac 0x112233445566 --
sourcetagged 3 --sourcevid 20 --streamhandle 100
```

4. Send frame from test center. The frame passes to stream filter index 1.

```
tsntool> qcisfiget --device eno0 --index 1
```

8.1.3.3.4 SGI stream gate list

Use the command below for this test:

```
cat > sgi1.txt << EOF
t0 0b -1 1000 0
t1 1b -1 1000 0
EOF
tsntool> qcisfiset --device eno0 --index 2 --gateid 2
tsntool> qcisgiset --device eno0 --index 2 --initgate 1 --gatelistfile sgi1.txt

#flooding frame size 64bytes at test center
tsntool> qcisfiget --device eno0 --index 2
```

Check the frames dropped and passed, they should be the same.

8.1.3.3.5 FMI test

Only send green color frames, set the test center speed to 10000000 bsp/s:

```
tsntool> qcisfiset --device eno0 --index 2 --gateid 2 --flowmeterid 2
tsntool> qcifmiset --device eno0 --index 2 --cm --cf --cbs 1500 --cir 5000000 --ebs 1500 --eir 5000000
```

The below setting shows the dropped frames:

```
tsntool> qcifmiset --device eno0 --index 2 --cm --cf --cbs 1500 --cir 5000000 --ebs 1500 --eir 2000000
```

To get information of color frame counters showing at application layer, use the code as in the below example:

```
tsntool> qcifmiget --device eno0 --index 2
=======================================================================
bytecount drop dr0_green dr1_green dr2_yellow remark_yellow dr3_red remark_red
1c89 0 4c 0 0 0 0 0
=======================================================================
index = 2
cir = c34c
cbs = 5dc
eir = 4c4b3c
ebs = 5dc
couple flag
color mode
```

8.1.3.4 Qbu test

Set the frame path from eno0 to external by linking enetc MAC0 - SWP0. Use the setup as shown in the following figure for the Qbu test.

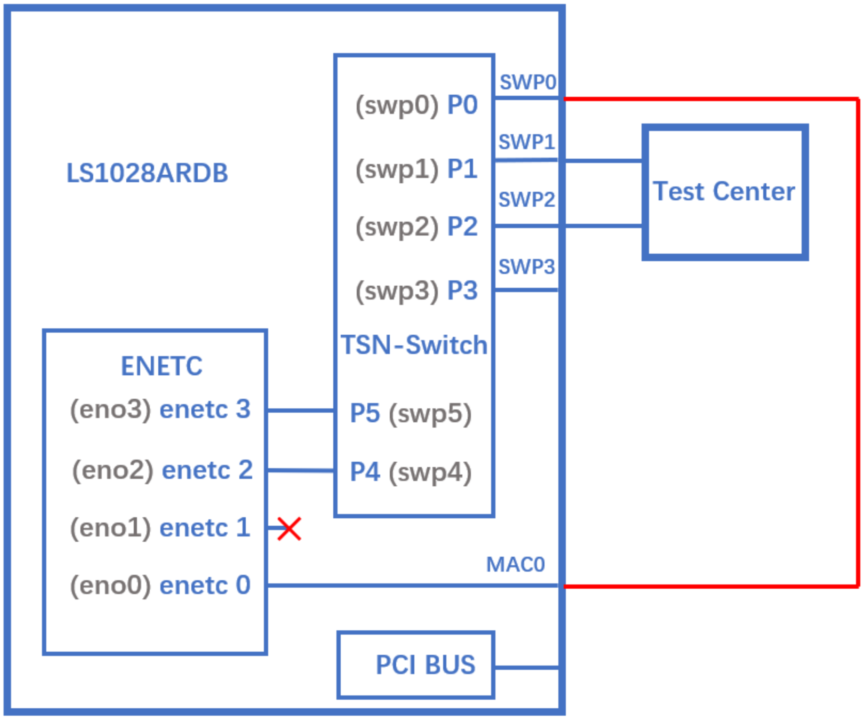


Figure 29. Qbu test

NOTE

0x11f10 Port MAC Merge Frame Assembly OK Count Register

```
0x11f18 Port MAC Merge Fragment Count TX Register (MAC_MERGE_MMFCTXR)
```

Before link the cable between ENETC port0 to SWP0, set up the switch up(refer the Switch configuration) and set IP for ENETC port0. To make sure linking the ENETC port0 to SWP0, use the steps below:

1. Make sure link speed is 1 Gbps by using the command:

```
ethtool eno0
```

2. If it is not 1Gbps, set it to 1 Gbps by using the command:

```
ethtool -s swp0 speed 1000 duplex full autoneg on
```

3. Set the switch to enable merge:

```
devmem 0x1fc100048 32 0x111 #DEV_GMII:MM_CONFIG:ENABLE_CONFIG
```

4. ENETC port setting set and frame preemption test

```
ip link set eno0 address 90:e2:ba:ff:ff:ff
tsntool> qbuset --device eno0 --preemptable 0xfe
./pktgen/pktgen_twoqueue.sh -i eno0 -q 0 -s 100 -n 20000 -m 90:e2:ba:ff:ff:ff
```

pktgen would fluding frames on TC0 and TC1.

5. Check the tx merge counter, if it has a non-zero value, it indicates that the Qbu is working.

```
tsntool> regtool 0 0x11f18
```

8.1.3.5 Qav test

The following figure illustrates the hardware setup diagram for the Qav test.

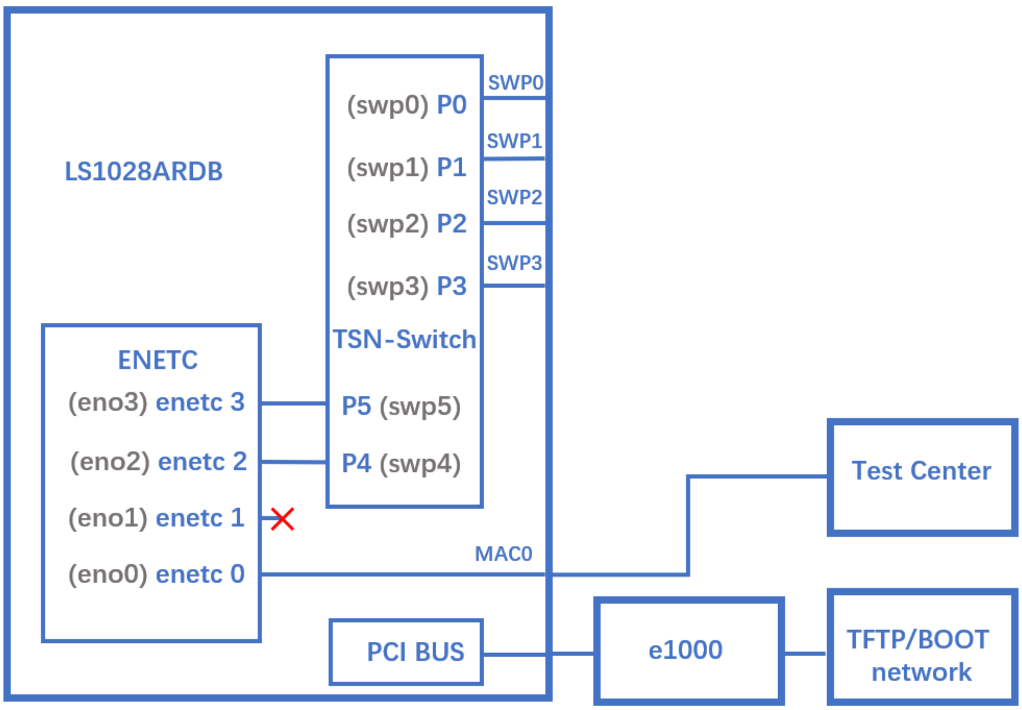


Figure 30. Qav test setup

1. Run the following commands:

```
cbsset --device eno0 --tc 7 --percentage 60
cbsset --device eno0 --tc 6 --percentage 20
```

2. Check each queue bandwidth

```
./pktgen/pktgen_sample01_simple.sh -i eno0 -q 7 -s 500 -n 300
```

wait seconds later to check result. It should get about 60% percentage line rate.

```
./pktgen/pktgen_sample01_simple.sh -i eno0 -q 6 -s 500 -n 300
```

Wait seconds later to check result. It should get about 20% percentage line rate.

8.1.4 Basic TSN configuration examples on the switch

The following sections describe examples for the basic configuration of TSN switch.

8.1.4.1 Switch configuration

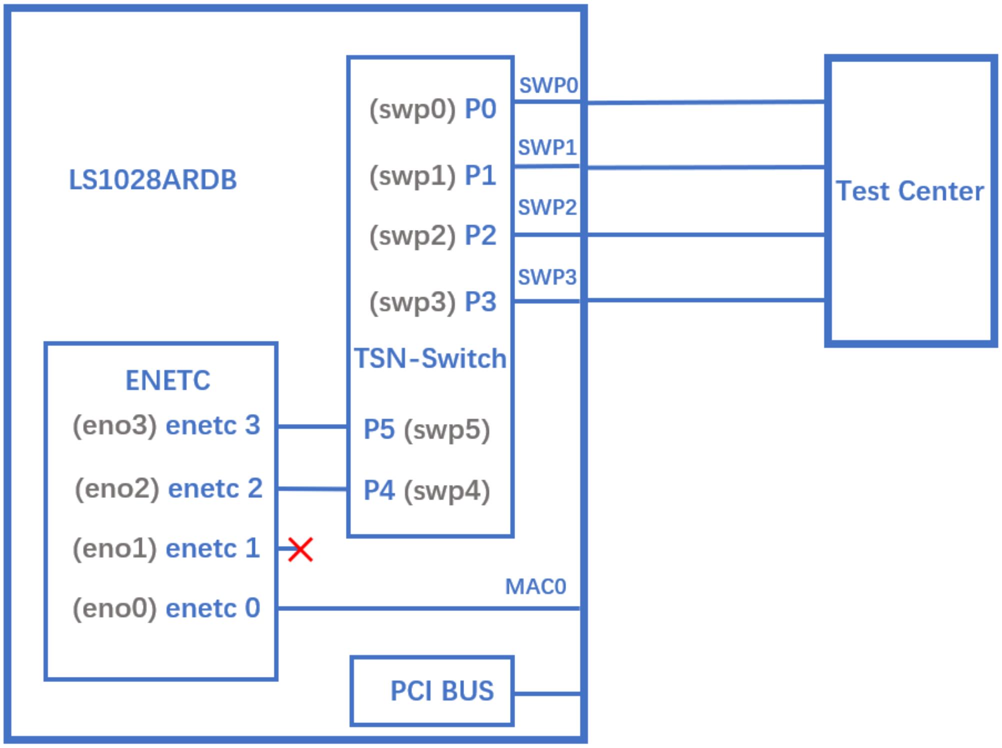


Figure 31. TSN switch configuration

Use the following commands for configuring the switch on LS1028ARDB:

```
ls /sys/bus/pci/devices/0000:00:00.5/net/
```

Get switch device interfaces: swp0 swp1 swp2 swp3 swp4 swp5>

```
ip link add name switch type bridge
ip link set switch up
ip link set swp0 master switch && ip link set swp0 up
ip link set swp1 master switch && ip link set swp1 up
ip link set swp2 master switch && ip link set swp2 up
ip link set swp3 master switch && ip link set swp3 up
ip link set swp4 master switch && ip link set swp4 up
ip link set swp5 master switch && ip link set swp5 up
```

8.1.4.2 Linuxptp test

To test 1588 synchronization on felix-switch interfaces, connect two boards back-to-back with switch interfaces. For example, swp0 to swp0. The Linux booting log is displayed below:

```
…
pps pps0: new PPS source ptp1
…
```

Check PTP clock and timestamping capability

```
# ethtool -T swp0
Time stamping parameters for swp0:
Capabilities:
 hardware-transmit  (SOF_TIMESTAMPING_TX_HARDWARE)
 hardware-receive   (SOF_TIMESTAMPING_RX_HARDWARE)
 hardware-raw-clock (SOF_TIMESTAMPING_RAW_HARDWARE)
PTP Hardware Clock: 1
Hardware Transmit Timestamp Modes:
  off   (HWTSTAMP_TX_OFF)
  on    (HWTSTAMP_TX_ON)
 Hardware Receive Filter Modes:
  none  (HWTSTAMP_FILTER_NONE)
  all   (HWTSTAMP_FILTER_ALL)
```

For 802.1AS testing, use the configuration file `gPTP.cfg` in linuxptp source. Run the below commands on the two boards instead.

```
# ptp4l -i swp0 -p /dev/ptp1 -f gPTP.cfg -m
```

8.1.4.3 Qbv test

The following figure describes the setup for Qbv test on LS1028ARDB.

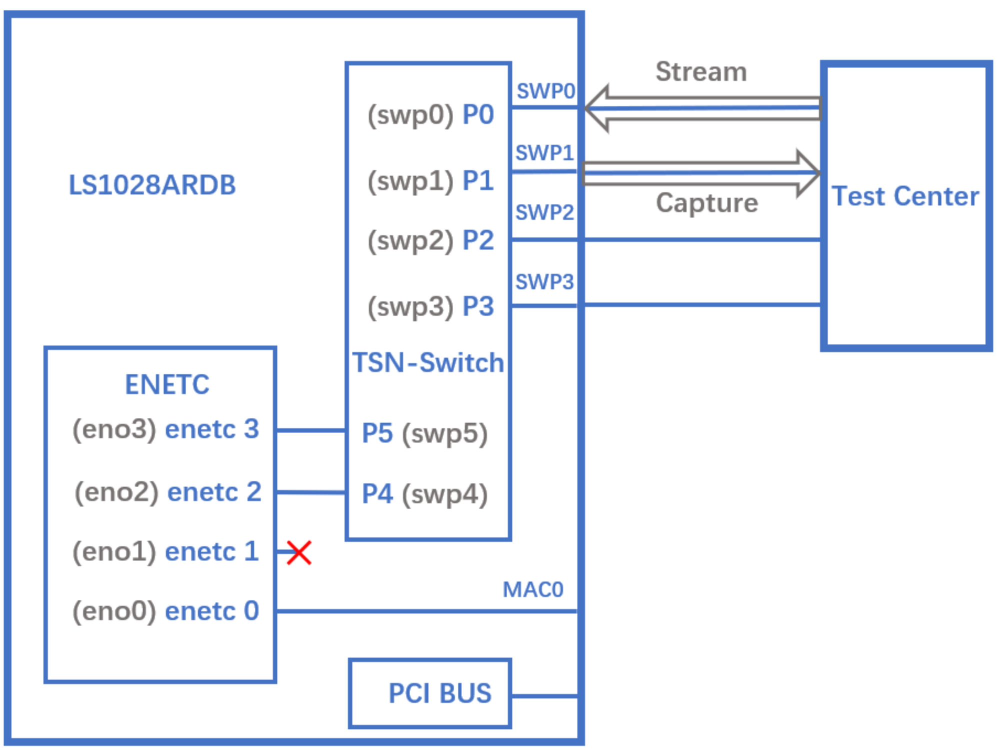


Figure 32. Qbv test

8.1.4.3.1 Closing basic gates

Use the set of commands below for basic gate closing.

```
echo "t0 00000000b 20000" > qbv0.txt
#Explaination:
# 'NUMBER'      :  t0
# 'GATE_VALUE'  :  00000000b
# 'TIME_LONG'   :  20000 ns

cp libtsn.so /lib
./tsntool
tsntool> verbose
tsntool> qbvset --device swp1 --entryfile ./qbv0.txt

#Send one broadcast frame to swp0 from TestCenter.
ethtool -S swp1
#Should not get any frame from swp1 on TestCenter.

echo "t0 11111111b 20000" > qbv0.txt
tsntool> qbvset --device swp1 --entryfile ./qbv0.txt

#Send one broadcast frame to swp0 on TestCenter.
ethtool -S swp1
#Should get one frame from swp1 on TestCenter.
```

8.1.4.3.2 Basetime test

For the basetime test, first get the current second time:

```
#Get current time:
tsntool> ptptool -g -d /dev/ptp1

#add some seconds, for example you get 200.666 time clock, then set 260.666 as result
tsntool> qbvset --device swp1 --entryfile ./qbv0.txt --basetime 260.666

#Send one broadcast frame to swp0 on the Test Center.
#Frame could not pass swp1 until time offset.
```

8.1.4.3.3 Qbv performance test

Use the following commands for the QBv performance test:

```
cat > qbv5.txt << EOF
t0 11111111b 1000000
t1 00000000b 1000000
EOF
qbvset --device swp1 --entryfile qbv5.txt
```

#Send 1G rate stream to swp0 on TestCenter.

#The stream would get about half line rate from swp1.

8.1.4.4 Qbu test

The figure below illustrates the setup for performing the Qbu test using the TSN switch.

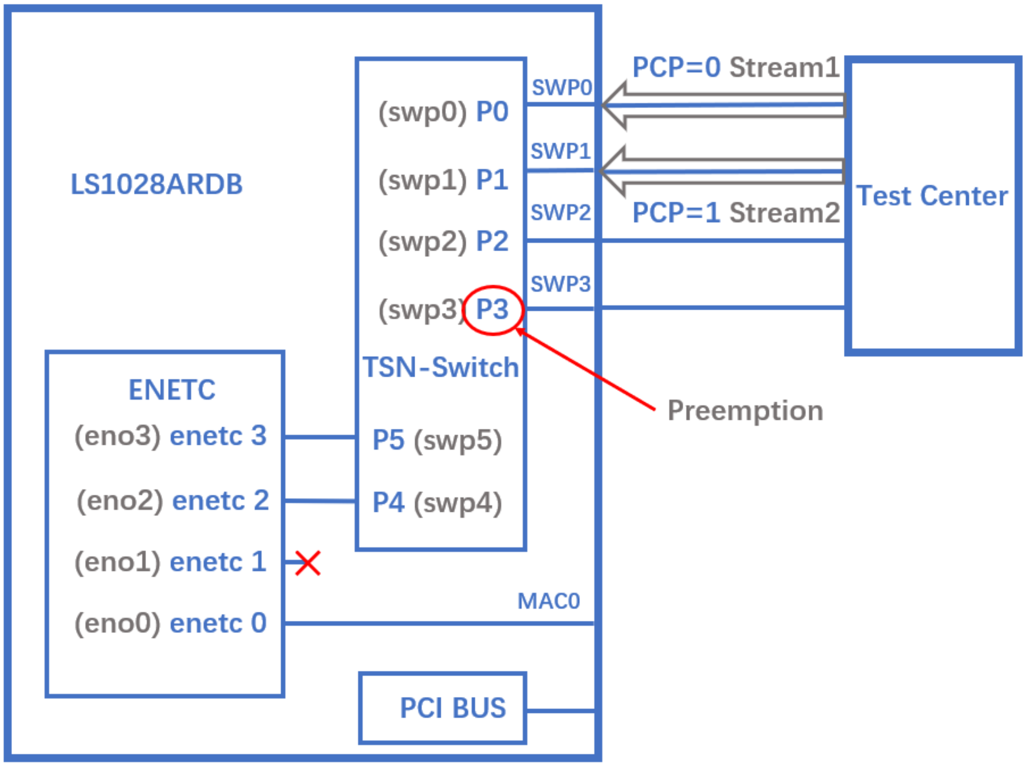


Figure 33. Qbu test on switch

1. Set queue 1 to be preemptable.

```
tsntool> qbuset --device swp3 --preemptable 0x02
```

2. Send two streams from TestCenter, then check the number of additional mPackets transmitted by PMAC:

```
devmem 0x1fc010e48 32 0x3 && devmem 0x1fc010280
```

8.1.4.5 Qci test cases

The figure below illustrates the Qci test case setup.

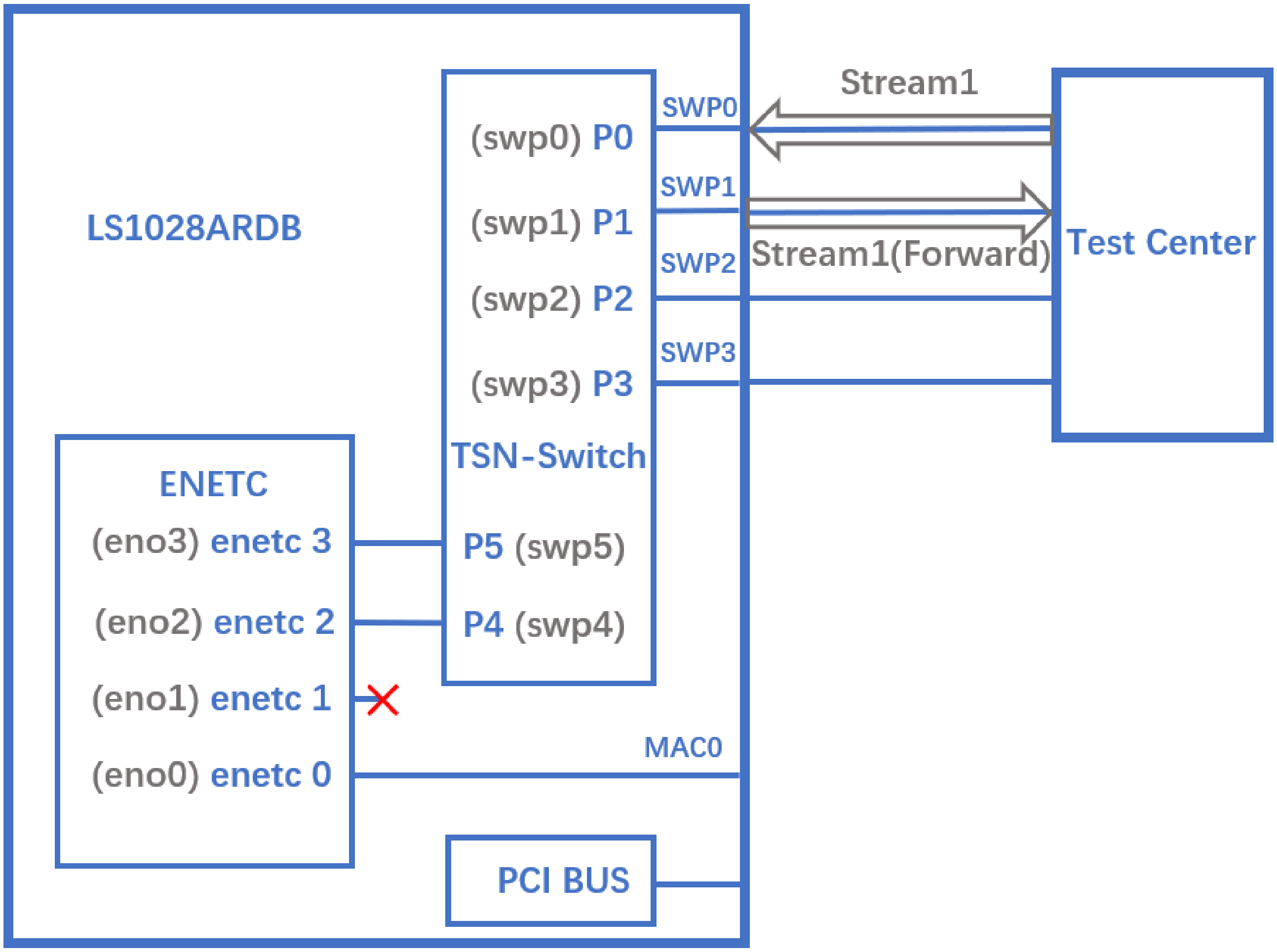


Figure 34. Qci test case

8.1.4.5.1 Stream identification

Use the following commands for stream identification:

1. Set a stream to `swp0` on TestCenter.
2. Edit the stream, set the destination MAC as: `00:01:83:fe:12:01, Vlan ID : 1`

```
tsntool> cbstreamidset --device swp1 --nullstreamid --index 1 --nulldmac  0x000183fe1201 --
nullvid 1 --streamhandle 1
```

Explanation:

- `device`: set the device port which is the stream forwarded to. If the {destmac, VID} is already learned by switch, switch will not care device port.
- `nulltagged`: switch only support nulltagged=1 mode, so there is no need to set it.
- `nullvid`: Use "bridge vlan show" to see the ingress VID of switch port.

```
tsntool> qcisfiset --device swp0 --index 1 --gateid 1 --priority 0 --flowmeterid 68
```

Explanation:

- `device`: can be any one of switch ports.
- `flowmeterid`: PSFP Policer id, ranges from 63 to 383.

3. Send one frame, then check the frames.

```
ethtool -S swp1
ethtool -S swp2
```

Only `swp1` can get the frame.

4. Use the following command to check and debug the stream identification status.

```
 qcisfiget --device swp0 --index 1
```

NOTE

The parameter **`streamhandle`** is the same as `index` in stream filter set, we use `streamhandle` in `cbstreamidset` to set a stream filter entry, and use `index` to disable it. Also, we use `index` in `cbstreamidget` to get this stream filter entry.

8.1.4.5.2 Stream gate control

1. Use the following commands for stream gate control:

```
echo "t0 1b 3 50000 200" > sgi.txt
tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --gatelistfile
sgi.txt --basetime 0x0
```

Explanation:

- 'device': can be any one of switch ports.
- 'index': gateid
- 'basetime' : It is the same as Qbv set.

2. Send one frame on TestCenter.

```
ethtool -S swp1
```

Note that the frame could pass, and green_prio_3 has increased.

3. Now run the following commands:

```
echo "t0 0b 3 50000 200" > sgi.txtx
tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --gatelistfile
sgi.txt --basetime 0x0
```

4. Next, send one frame on TestCenter.

```
ethtool -S swp1
```

Note that the frame could not pass.

8.1.4.5.3 SFI maxSDU test

Use the following command to run this test:

```
tsntool> qcisfiset --device swp0 --index 1 --gateid 1 --priority 0 --flowmeterid 68 --maxsdu 200
```

Now, send one frame (frame size > 200) on TestCenter.

```
ethtool -S swp1
```

You can observe that the frame could not pass.

8.1.4.5.4 FMI test

Use the following set of commands for the FMI test.

1. Run the command:

   ```
   tsntool> qcifmiset --device swp0 --index 68 --cir 100000 --cbs 4000 --ebs 4000 --eir 100000
   ```

   NOTE
   - The 'device' in above command can be any one of the switch ports.
   - The index of `qcifmiset` must be the same as flowmeterid of `qcisfiset`.

2. Now, send one stream (rate = 100M) on TestCenter.

   ```
   ethtool -S swp0
   ```

   Note that all frames pass and get all green frames.

3. Now, send one stream (rate = 200M) on TestCenter.

   ```
   ethtool -S swp0
   ```

   Observe that all frames pass and get green and yellow frames.

4. Send one stream (rate = 300M) on TestCenter.

   ```
   ethtool -S swp0
   ```

   Note that not all frames could pass and get green, yellow, and red frames.

5. Map the CFI value of VLan to dp value on port 0 to recognize yellow frames.

   ```
   tsntool> pcpmap --device swp0 --enable
   ```

6. Send one yellow stream (rate = 100M) on TestCenter.

   ```
   ethtool -S swp0
   ```

   All frames pass and get all yellow frames.

7. Send one yellow stream (rate = 200M) on TestCenter.

   ```
   ethtool -S swp0
   ```

   Note that not all frames could pass and get yellow and red frames.

8. Test cf mode.

   ```
   tsntool> qcifmiset --device swp0 --index 68 --cir 100000 --cbs 4000 --ebs 4000 --eir 100000 --cf
   ```

9. Send one yellow stream (rate = 200M) on TestCenter.

   ```
   ethtool -S swp0
   ```

All frames pass and get all yellow frames (use CIR as well as EIR).

10. Send one yellow stream (rate = 300M) on TestCenter.

```
ethtool -S swp0
```

NOTE
Note that not all frames could pass and get yellow and red frames.

8.1.4.6 Qav test case

The below figure illustrates the Qav test case setup.

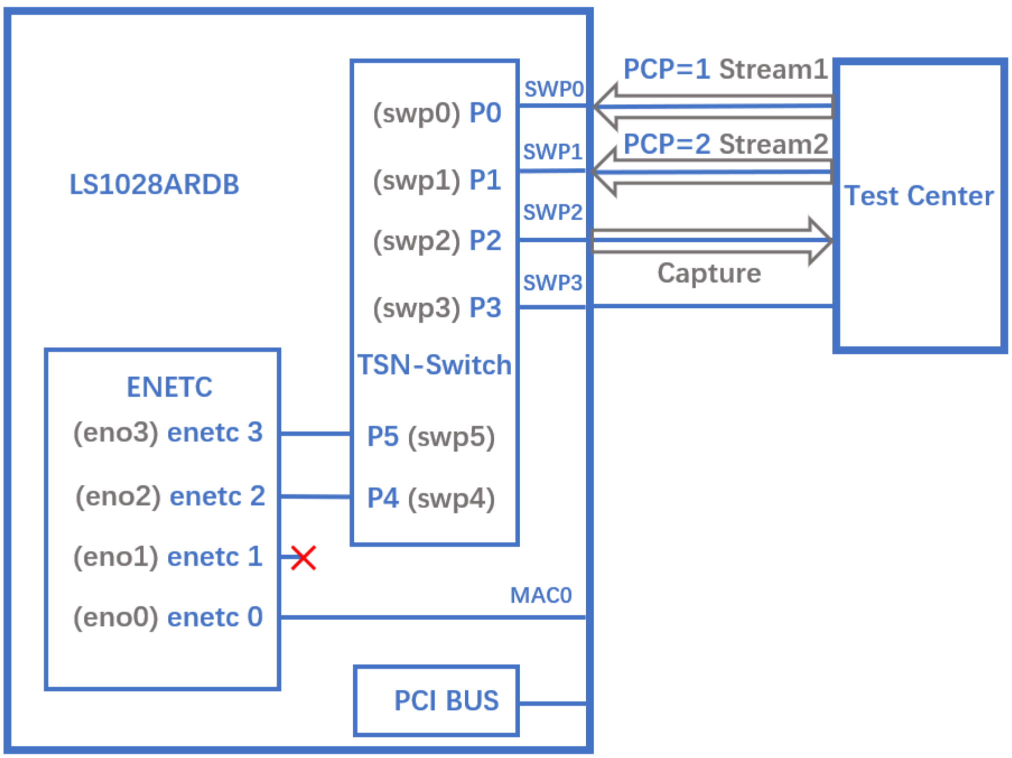


Figure 35. Qav test case

1. Set the percentage of two traffic classes:

```
tsntool> cbsset --device swp2 --tc 1 --percentage 20
tsntool> cbsset --device swp2 --tc 2 --percentage 40
```

2. Send two streams from Test center, then check the frames count.

```
ethtool -S swp2
```

Note that the frame count of queue1 is half of queue2.

NOTE
Stream rate must lager than bandwidth limited of queue.

3. Capture frames on swp2 on TestCenter.

```
# The Get Frame sequence is: (PCP=1), (PCP=2), (PCP=2), (PCP=1), (PCP=2), (PCP=2),…
```

8.1.4.7 Seamless redundancy test case

The following figure describes the test setup for the seamless redundancy test case.

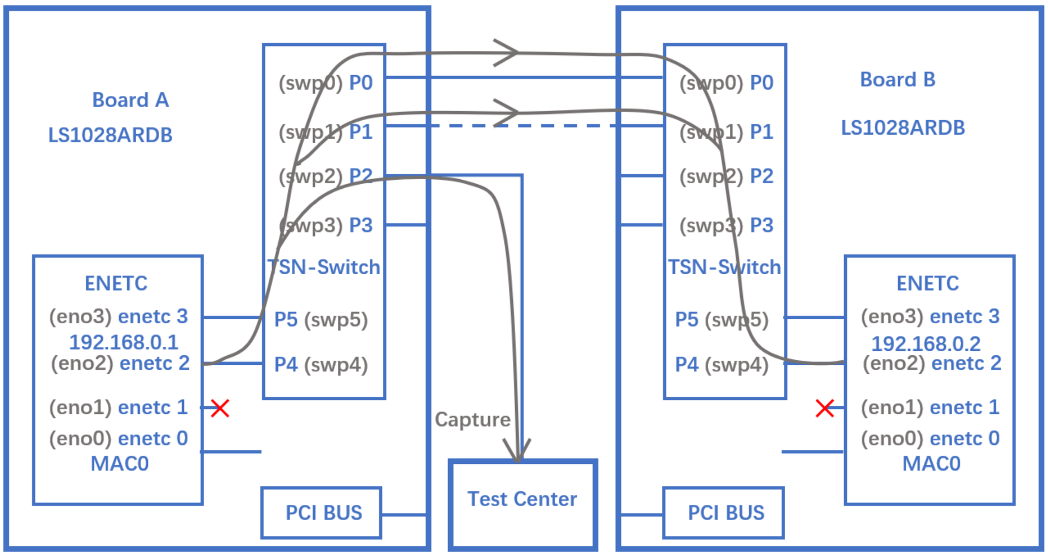


Figure 36. Seamless redundancy test

8.1.4.7.1 Sequence Generator test

Use the following set of commands for the 'Sequence Generator' test.

1. Configure switch ports to be forward mode.

   On board A:

```
ip link add name switch type bridge
ip link set switch up
ip link set swp0 master switch && ip link set swp0 up
ip link set swp1 up
ip link set swp2 master switch && ip link set swp2 up
ip link set swp3 master switch && ip link set swp3 up
ip link set swp4 master switch && ip link set swp4 up
ip link set swp5 master switch && ip link set swp5 up
```

   On board B

```
ip link add name switch type bridge
ip link set switch up
ip link set swp0 master switch && ip link set swp0 up
ip link set swp1 master switch && ip link set swp1 up
ip link set swp2 master switch && ip link set swp2 up
ip link set swp3 master switch && ip link set swp3 up
ip link set swp4 master switch && ip link set swp4 up
ip link set swp5 master switch && ip link set swp5 up
```

2. Use the following commands on board B:

```
ifconfig eno2 192.168.0.2 up
ifconfig eno2
```

Get the MAC address of eno2: `7E:A8:8C:9B:41:DD`

3. Use the following commands on board A:

```
ifconfig eno2 192.168.0.1 up
ping 192.168.0.2
```

The network is connected now.

4. On board A, run the commands:

```
tsntool> cbstreamidset --device swp0 --nullstreamid --nulldmac 0x7EA88C9B41DD --nullvid 1 --
streamhandle 1
tsntool> cbgen --device swp0 --index 1 --iport_mask 0x10 --split_mask 0x07 --seq_len 16 --
seq_num 2048
```

In the command above,

- `device`: can be any one of switch ports.
- `index:` value is the same as streamhandle of cbstreamidset.

5. Capture frames on swp2 on TestCenter.
6. Now run the command below:

```
ping 192.168.0.2
```

We can get frames from swp2 on TestCenter, each frame adds the sequence number: 23450801, 23450802, 23450803…

7. On board B, run the below command:

```
tcpdump -i eno2 -w eno2.pcap
```

Users can also get a copy of frames from eno2 on board B, which are transmitted through swp0 of board A.

8.1.4.7.2 Sequence Recover test

Use the following steps for the **Sequence Recover** test:

1. On board B, run the following commands:

```
tsntool> cbstreamidset --device swp4  --nullstreamid --nulldmac 0x7EA88C9B41DD --nullvid 1  --
streamhandle 1
tsntool> cbrec --device swp0 --index 1 --seq_len 16 --his_len 31 --rtag_pop_en
tcpdump -i eno2 -w eno2.pcap
```

In the commands mentioned above:

- `device`: can be any one of switch ports.
- `index`: value is the same as `streamhandle` of `cbstreamidset`.

2. On board A, run the command:

```
ping 192.168.0.2
```

Then on board B, we can get a frame without sequence tags from board A.

3. Connect `swp1` of board A with `swp1` of board B.

4. On board B, run the following commands:

```
tcpdump -i eno2 -w eno2.pcap
```

5. On board A, run the command:

```
ping 192.168.0.2
```

   On board B, we can get only one frame without sequence tag, another frame from `swp0` or `swp1` is dropped.

   The index of `cbrec` must be the same as streamhandle of `cbstreamidset`.

8.1.4.8 TSN stream identification

TSN module uses QoS class to identify and control streams. There are three ways to identify the stream to different QoS class. These are explained in the following sections.

8.1.4.8.1 Stream identification based on PCP value of Vlan tag

The default QoS class is based on PCP of Vlan tag for a frame. If there is no Vlan tag for a frame, the default QoS class is 0.

Set the PCP value on TestCenter.

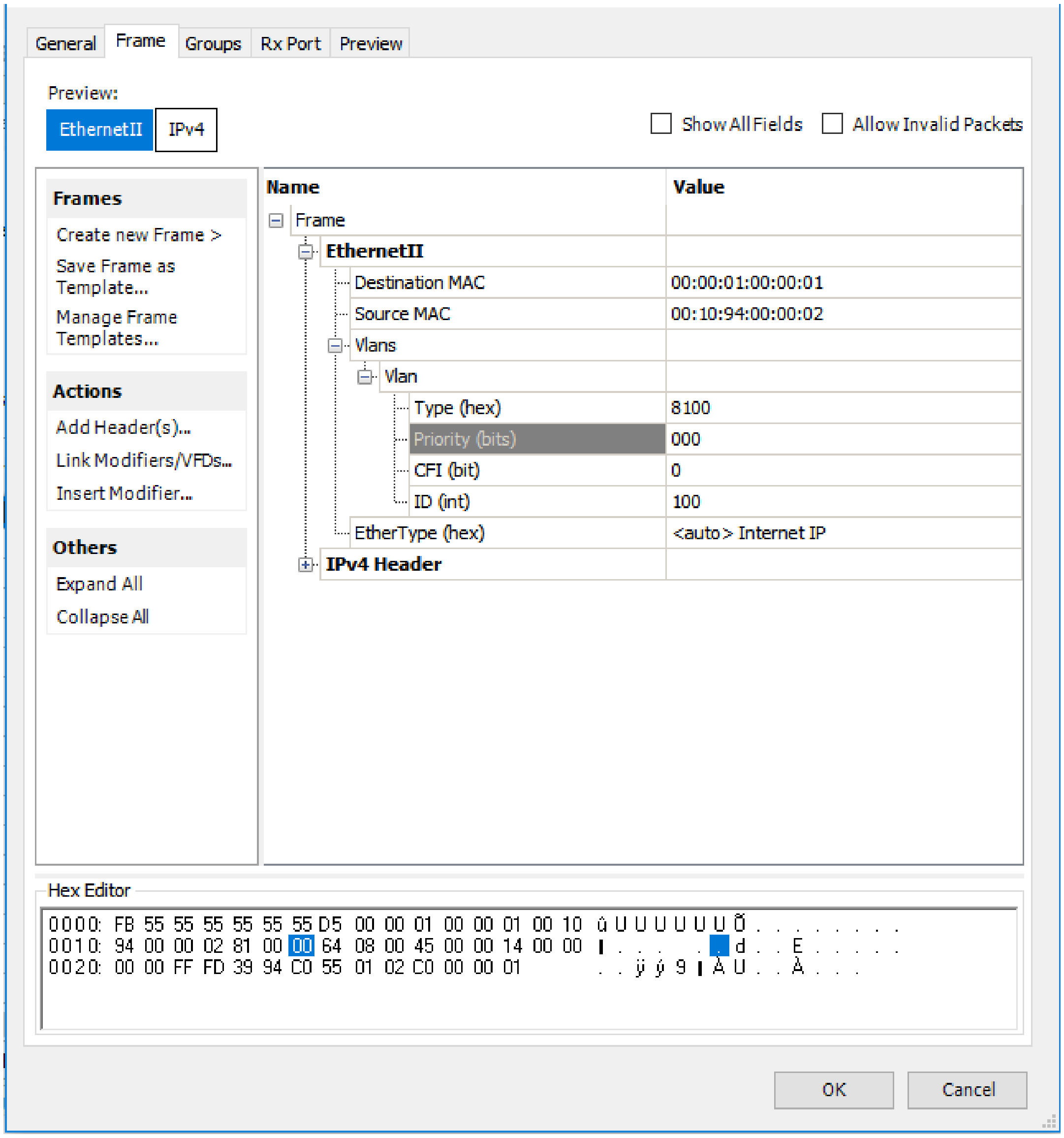


Figure 37. Using PCP value of Vlan tag

8.1.4.8.2 Based on DSCP of ToS tag

Use the below steps to identify stream based on DSCP value of ToS tag.

1. Map the DSCP value to a specific QoS class using the command below:

```
tsntool> dscpset --device swp0 --index 1 --cos 1 --dpl 0
```

Explanation:

- `index`: DSCP value of stream, 0-63.
- `cos`: QoS class which is mapped to.
- `dpl`: Drop level which is mapped to.

2. Set the DSCP value on TestCenter. DSCP value is the first six bits of ToS in IP header, set the DSCP value on TestCenter as shown in the following figure.

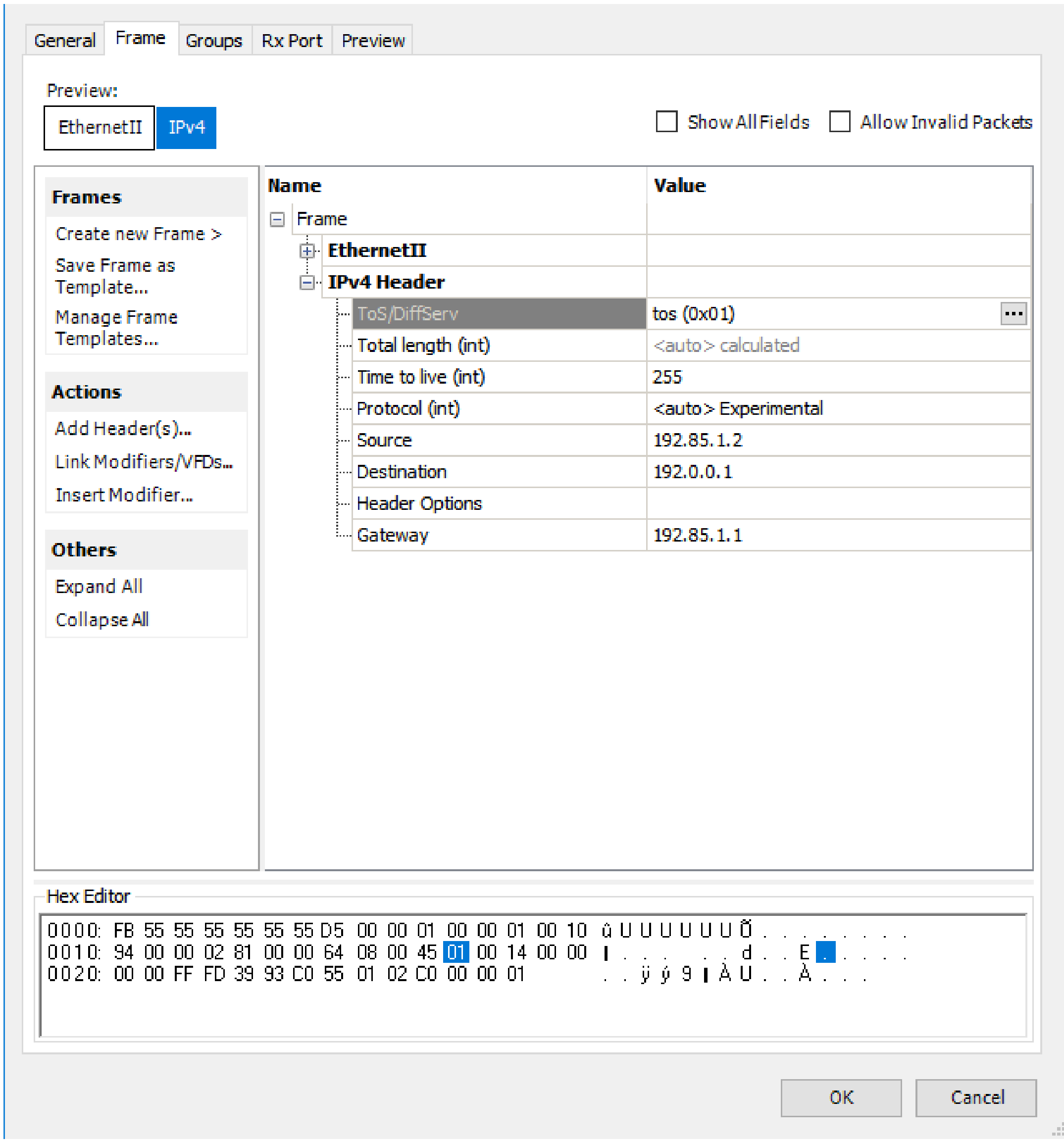


Figure 38. Setting DSCP value on TestCenter

8.1.4.8.3 Based on qci stream identification

The following steps describe how to use qci to identify the stream and set it to a QoS class.

1. Identify a stream.

```
tsntool> cbstreamidset --device swp1 --nullstreamid --nulldmac 0x000183fe1201 --nullvid 1 --
streamhandle 1
tsntool> qcisfiset --device swp0 --index 1 --gateid 1 --flowmeterid 68
```

2. Set to Qos class 3 by using stream gate control.

```
echo "t0 1b 3 50000 200" > sgi.txt
tsntool> qcisgiset --device swp0 --enable --index 1 --initgate 1 --initipv 0 --gatelistfile sgi.txt
```

8.1.4.9 ACL test

The access-control-list is using “tc flower” command to set the filter and actions. Following keys and actions are supported on LS1028a:

keys:
vlan_id
vlan_prio
dst_mac/src_mac for non IP frames
dst_ip/src_ip
dst_port/src_port

actions:
trap
drop
police

Using following commands to set, get and delete ACL rules:

```
tc qdisc add dev swp0 ingress
tc filter add dev swp0 parent ffff: protocol [ip/802.1Q] flower skip_sw [keys] action [actions]
tc filter list dev swp0 parent ffff:
tc filter del dev swp0 parent ffff: pref [pref_id]
```

There are two ACL use cases for testing:

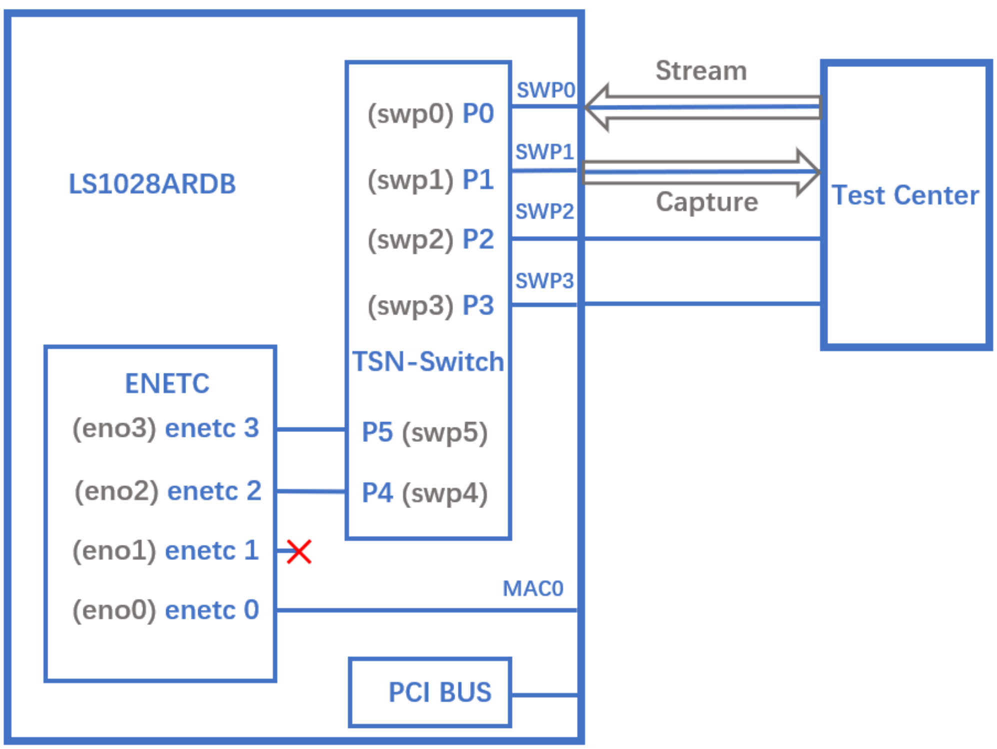


Figure 39. ACL test

1. Drop all frames from source IP 192.168.2.1.

```
tc qdisc add dev swp0 ingress
tc filter add dev swp0 parent ffff: protocol ip flower skip_sw src_ip 192.168.2.1 action drop
```

Set source IP as 192.168.2.1 and send ip package from testcenter, package will be dropped on swp0.

2. Limit bandwidth of HTTP streams to 10Mbps.

```
tc filter add dev eth3 parent ffff: protocol ip flower skip_sw dst_port 80 action police rate
10Mbit burst 10000
```

Send TCP package and set destination port as 80 on testcenter, set the stream bandwidth to 1Gbps, we can get a 10Mbps stream rate.

8.1.5 Netconf usage on LS1028ARDB

Netopeer is a set of NETCONF tools built on the libnetconf library. sysrepo-tsn (https://github.com/openil/sysrepo-tsn) helps to configure TSN features, including Qbv, Qbu, Qci, and stream identification via network, without logging in to device.

For details of configuring TSN features on LS1028ARDB, please refer to NETCONF/YANG).

8.2 Using TSN features on LS1021ATSN board

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.

8.2.1 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

8.2.2 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.

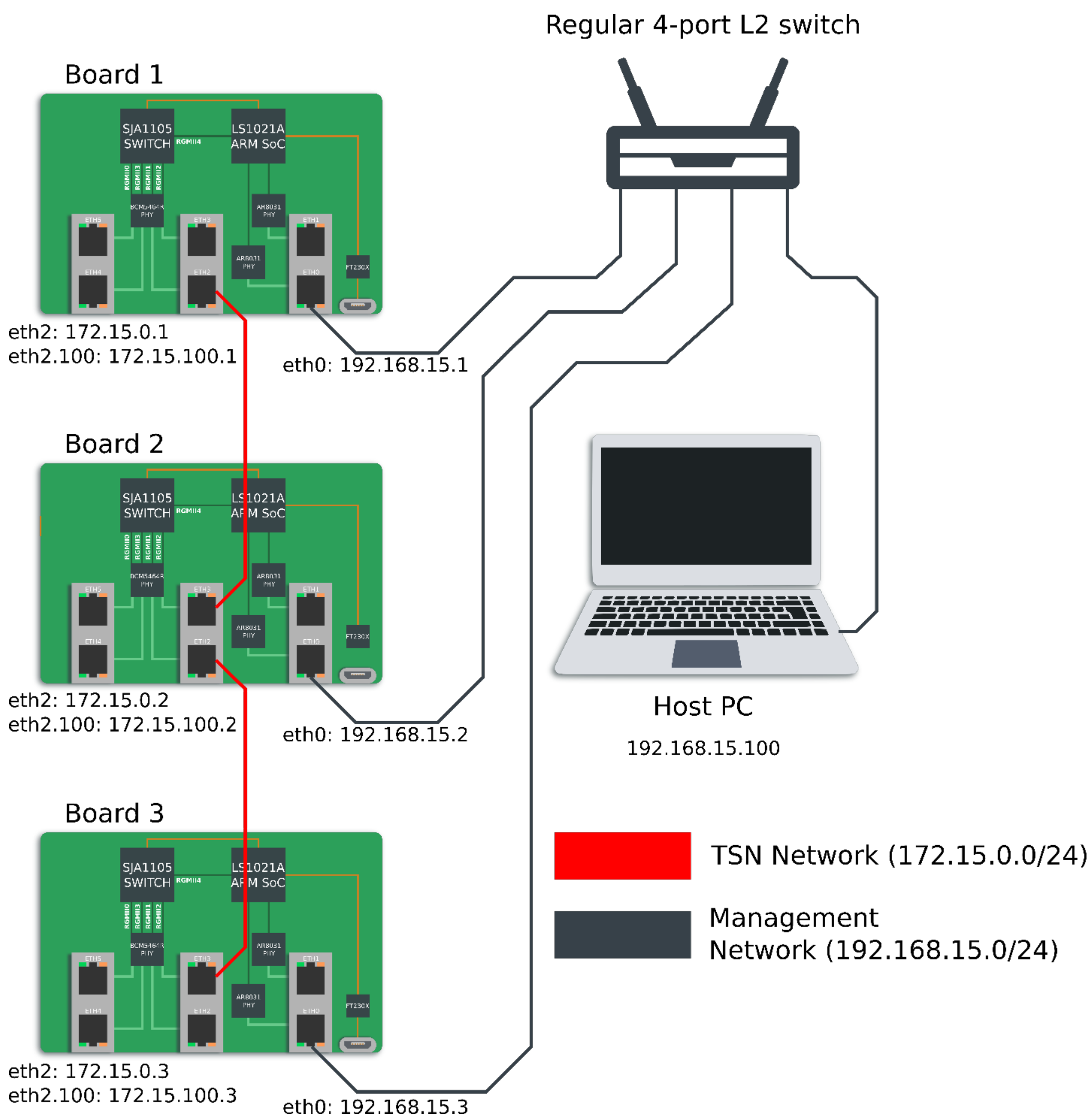


Figure 40. Topology of the demo network

8.2.3 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*.

8.2.4 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:

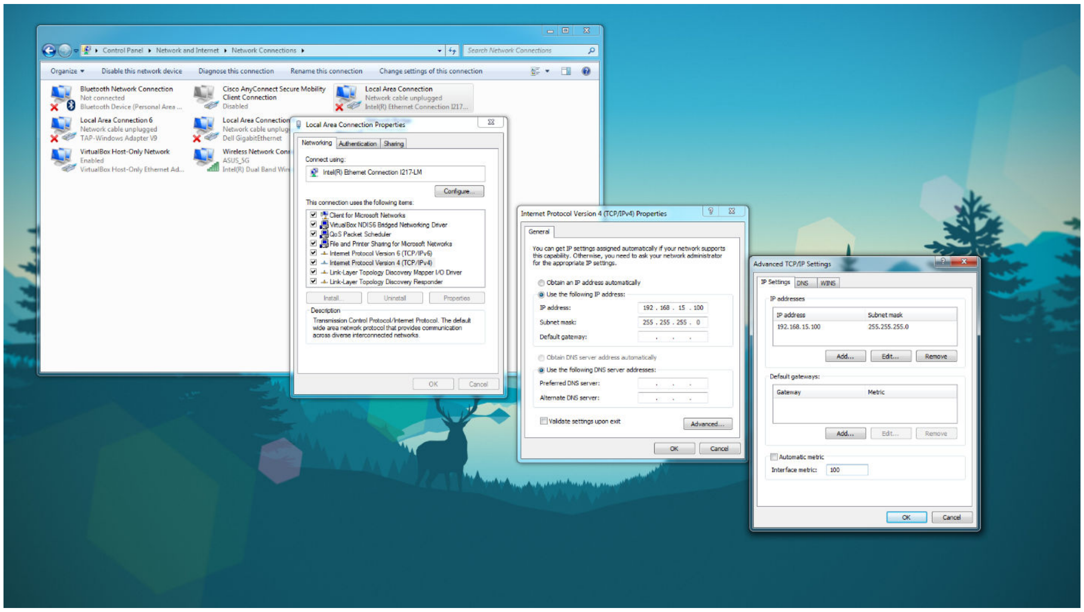


Figure 41. Windows network settings

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.

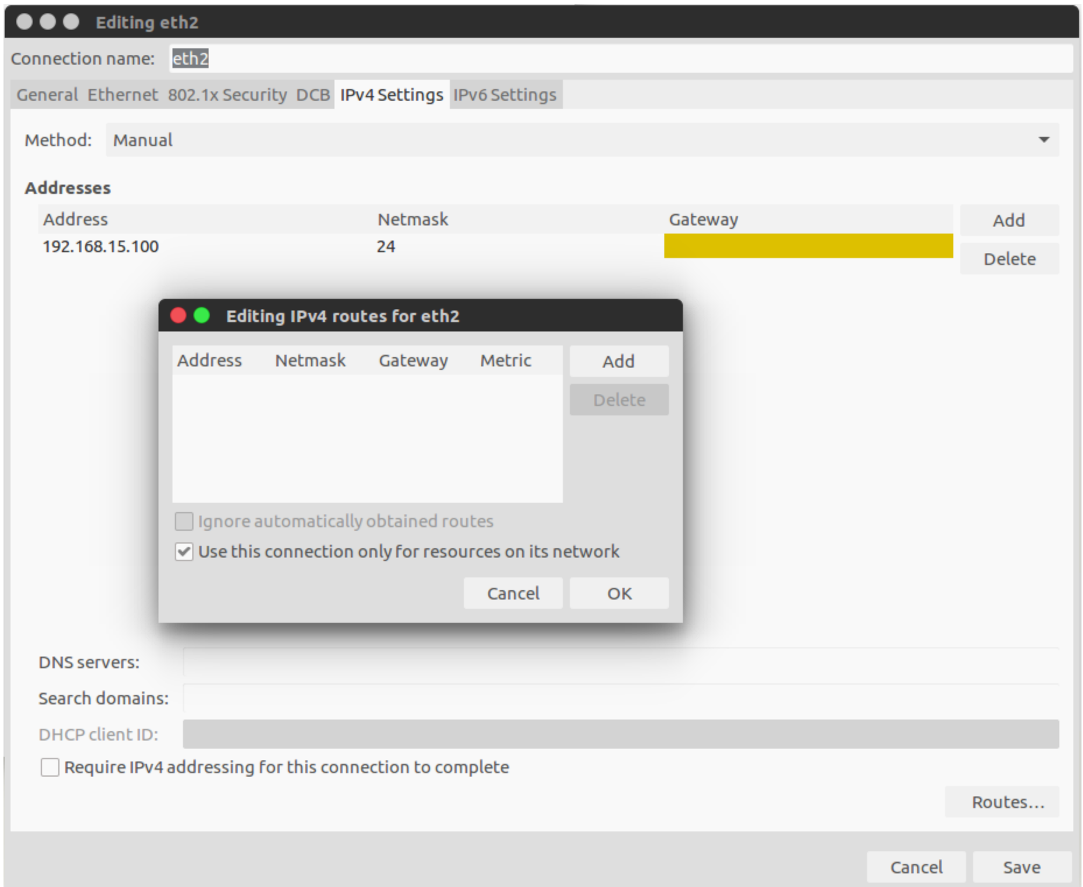


Figure 42. Ubuntu network settings (GNOME NetworkManager)

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 43. MobaXTerm session settings

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

Figure 44. Login shell to OpenIL on Board 1 in MobaXTerm

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

8.2.5 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

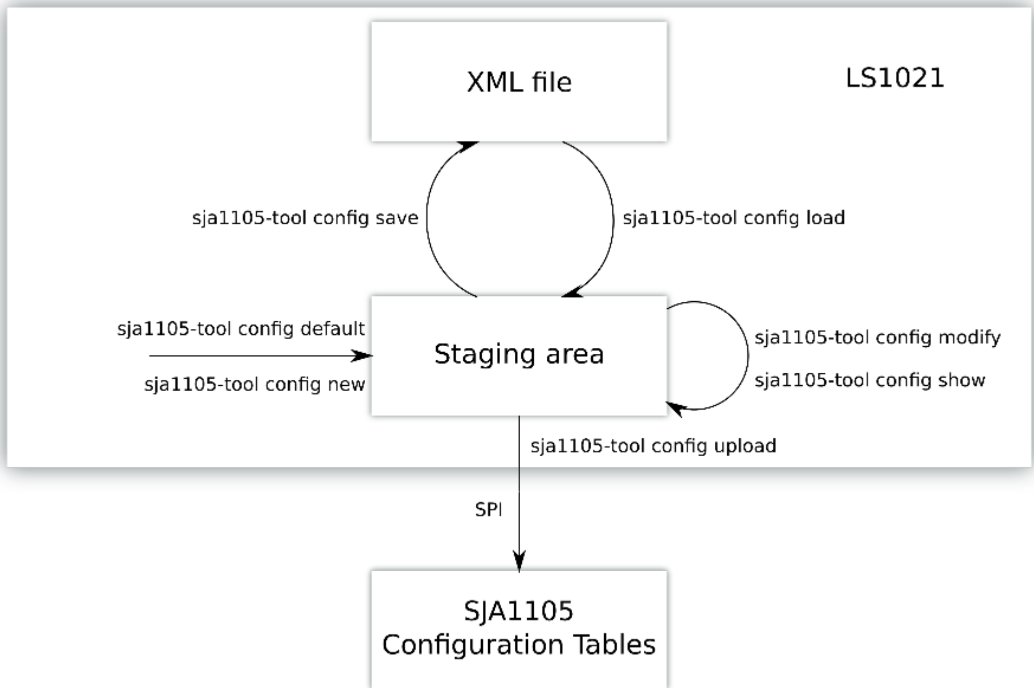
8.2.6 Managing configurations with the sja1105-tool

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 45. 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
```

8.2.6.1 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.

8.2.7 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.

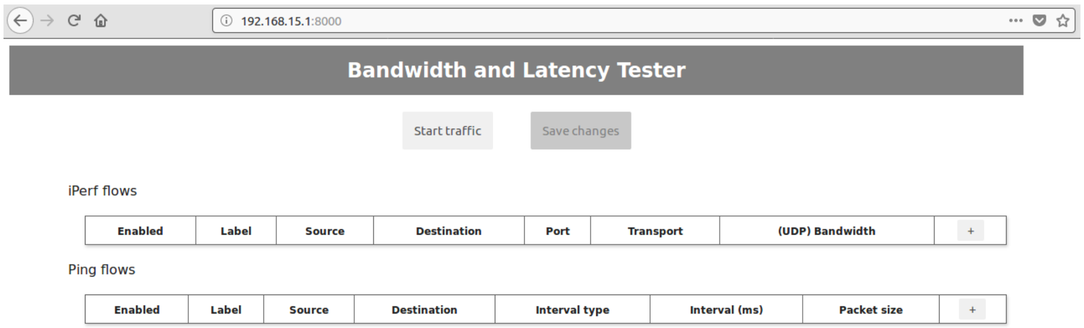


Figure 46. Latency and bandwidth tester default interface

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.`

8.2.8 Rate limiting demo

8.2.8.1 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).

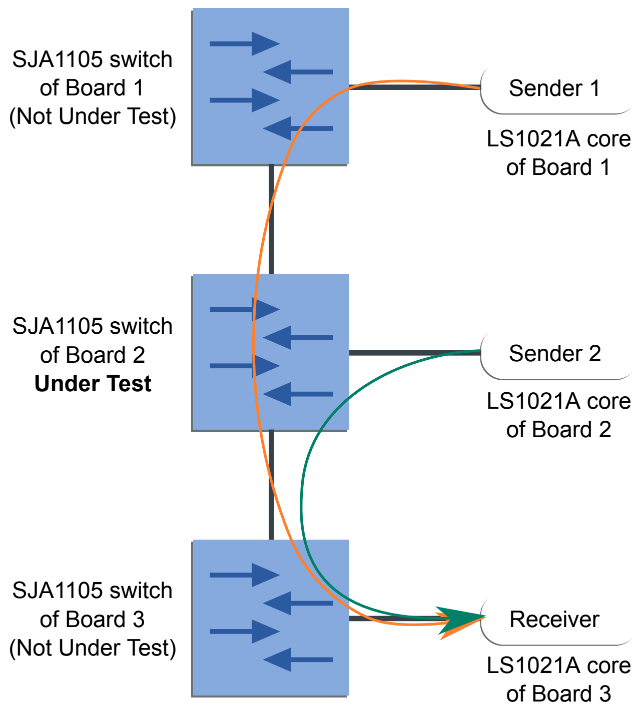


Figure 47. Connections of the three hosts and their roles as traffic senders/receivers

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

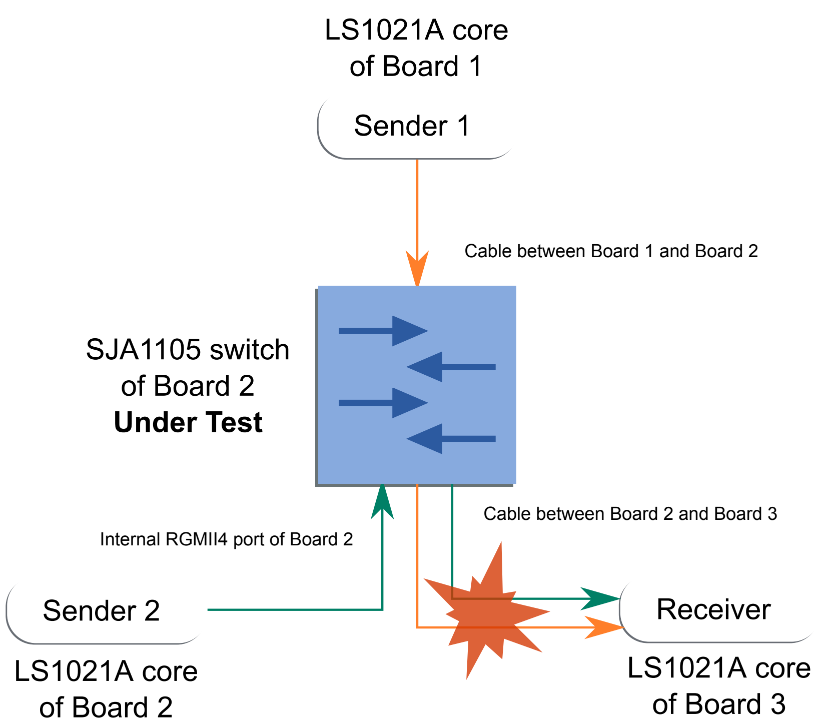


Figure 48. Simplified traffic flows for the rate limiting use case

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.

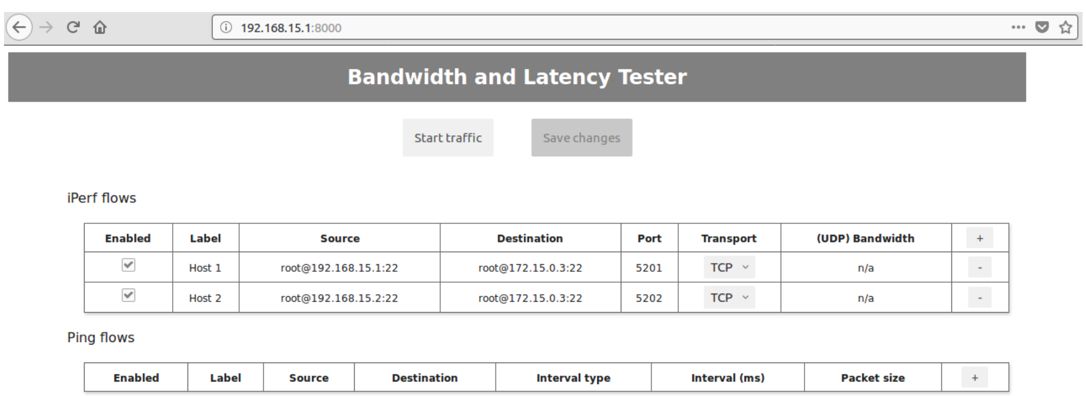
8.2.8.2 Objectives

- 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

8.2.8.3 Latency and bandwidth tester configuration

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

Figure 49. 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).

8.2.8.4 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.

8.2.8.5 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
```

8.2.8.5.1 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
```

8.2.8.5.2 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.

8.2.8.5.3 Queuing diagram

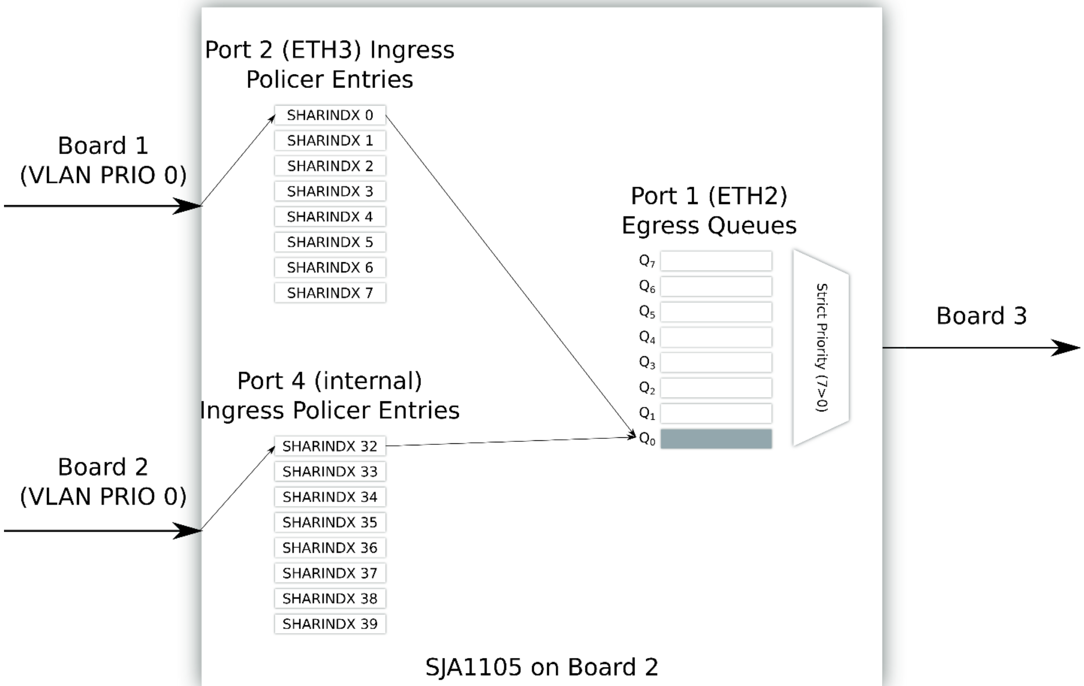


Figure 50. Queuing diagram

/>

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

8.2.8.5.4 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

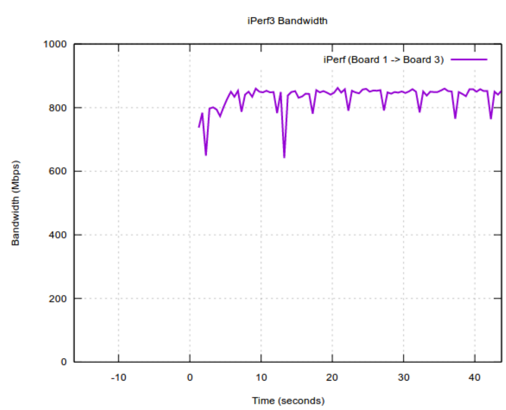


Figure 51. Standard configuration: Flow 1 run by its own

- Only flow 2 enabled

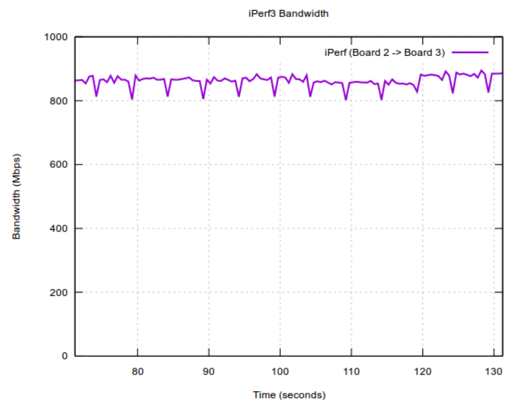


Figure 52. Standard configuration: Flow 2 run by its own

- Both flows enabled

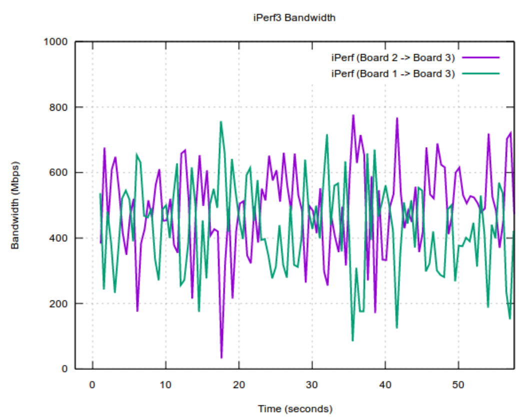


Figure 53. Standard configuration: Both flows run at the same time

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).

8.2.8.6 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.

8.2.8.6.1 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.

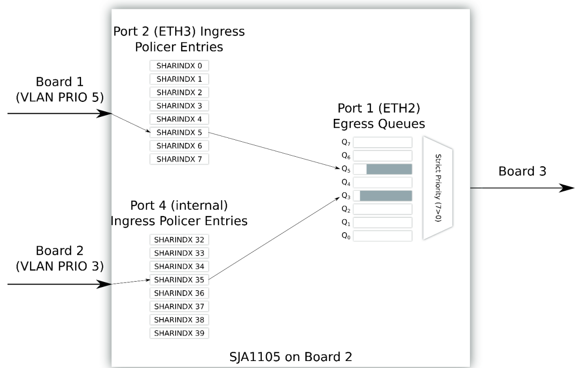


Figure 54. Prioritizing configuration

8.2.8.6.2 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

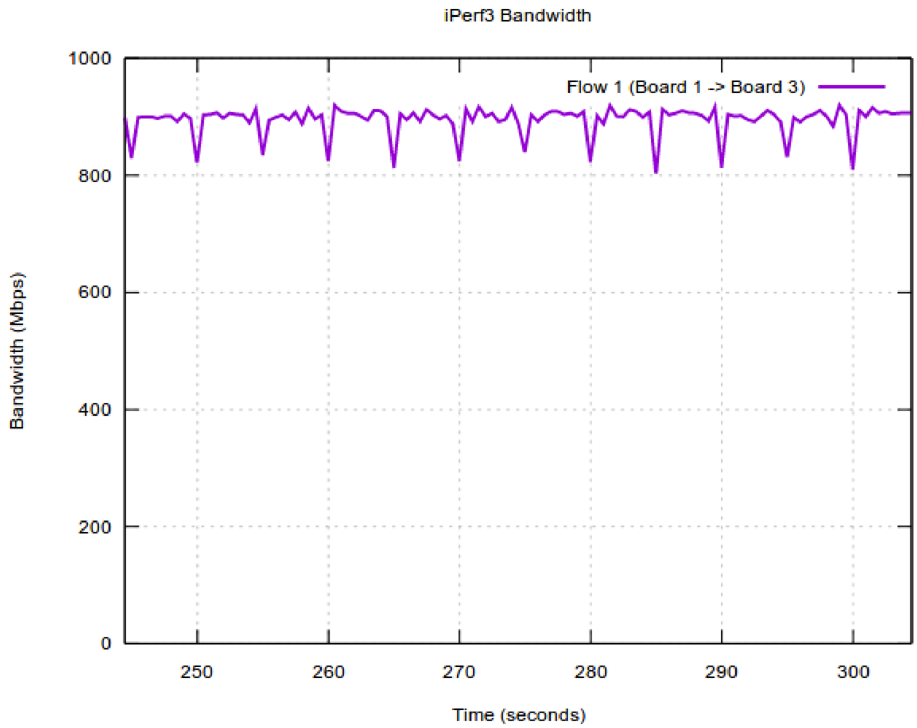


Figure 55. Prioritizing configuration: Flow 1 run by its own

- Flow 2 run by its own

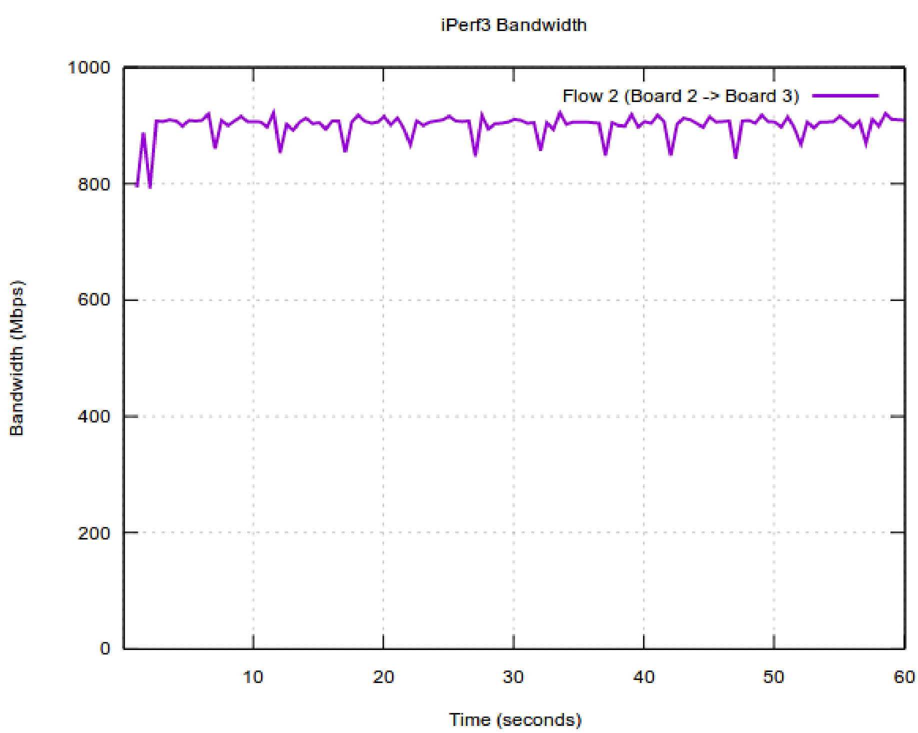


Figure 56. Prioritizing configuration: Flow 2 run by its own

- Both flows run at the same time

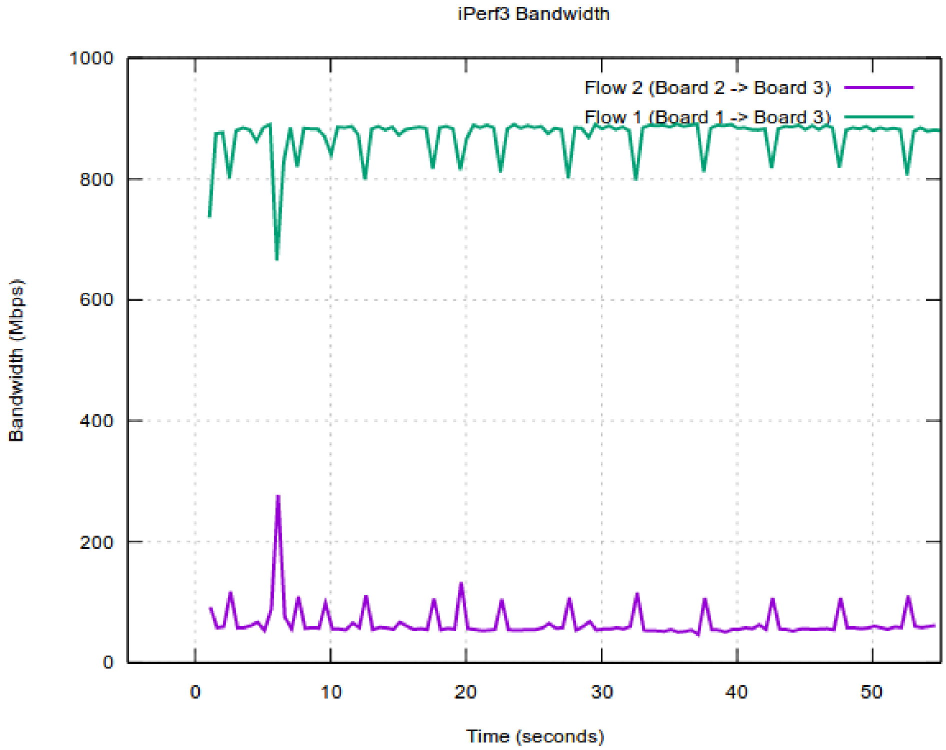


Figure 57. Prioritizing configuration: both flows run at the same time

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.

8.2.8.7 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"
```

8.2.8.7.1 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.

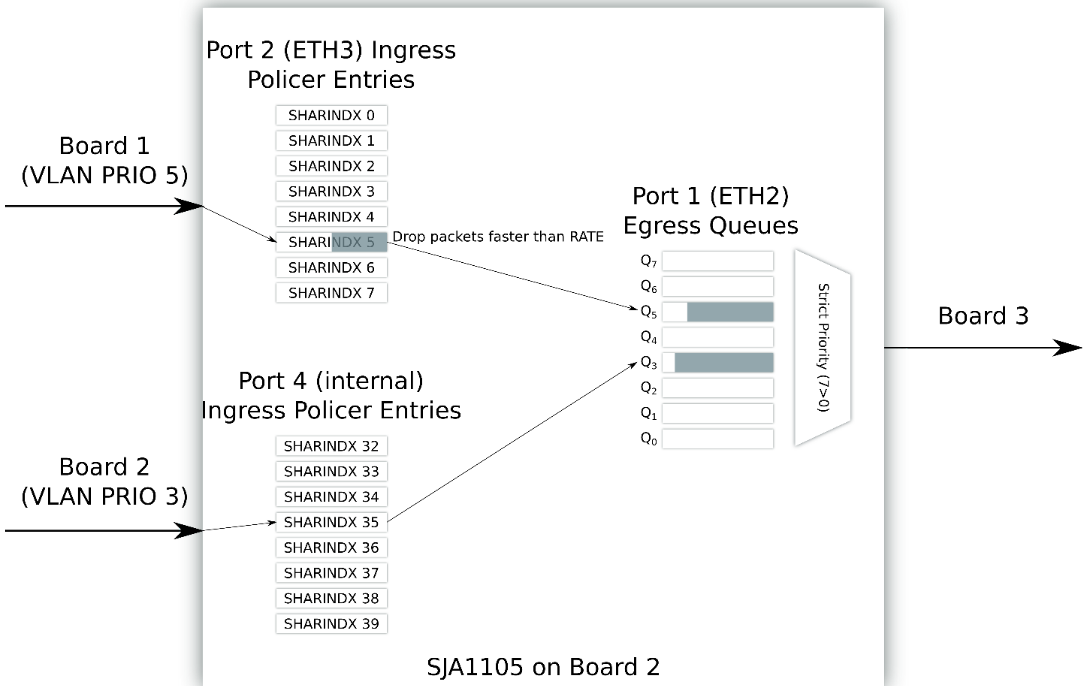


Figure 58. Queuing diagram for policing configuration

.

8.2.8.7.2 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

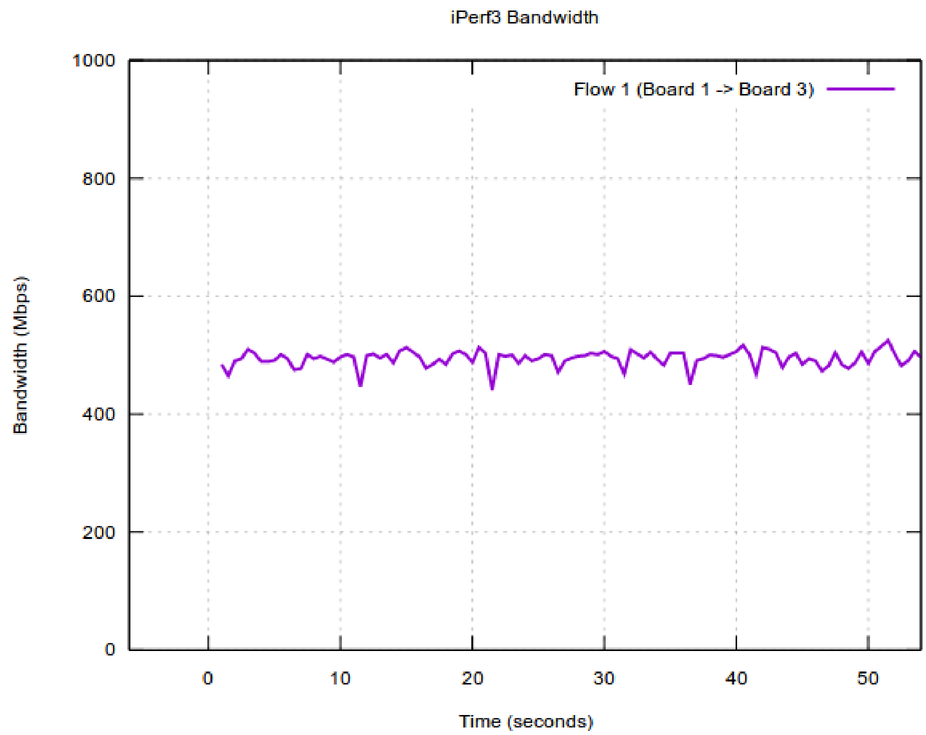


Figure 59. Policing configuration: Flow 1 run by its own

- Flow 2 run by its own

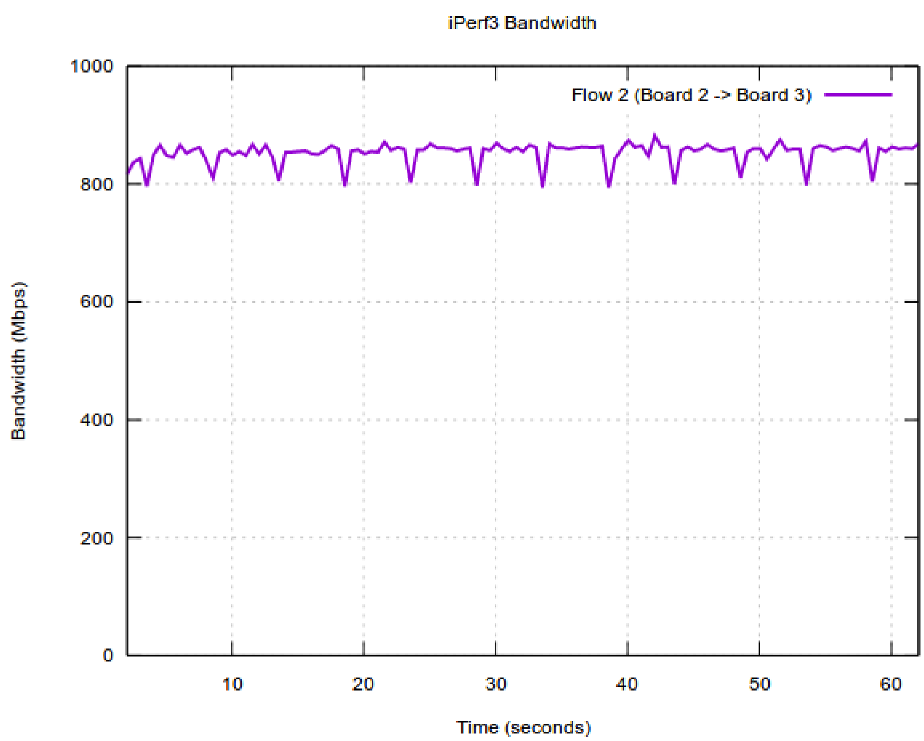


Figure 60. Policing configuration: Flow 2 run by its own

- Both flows run at the same time

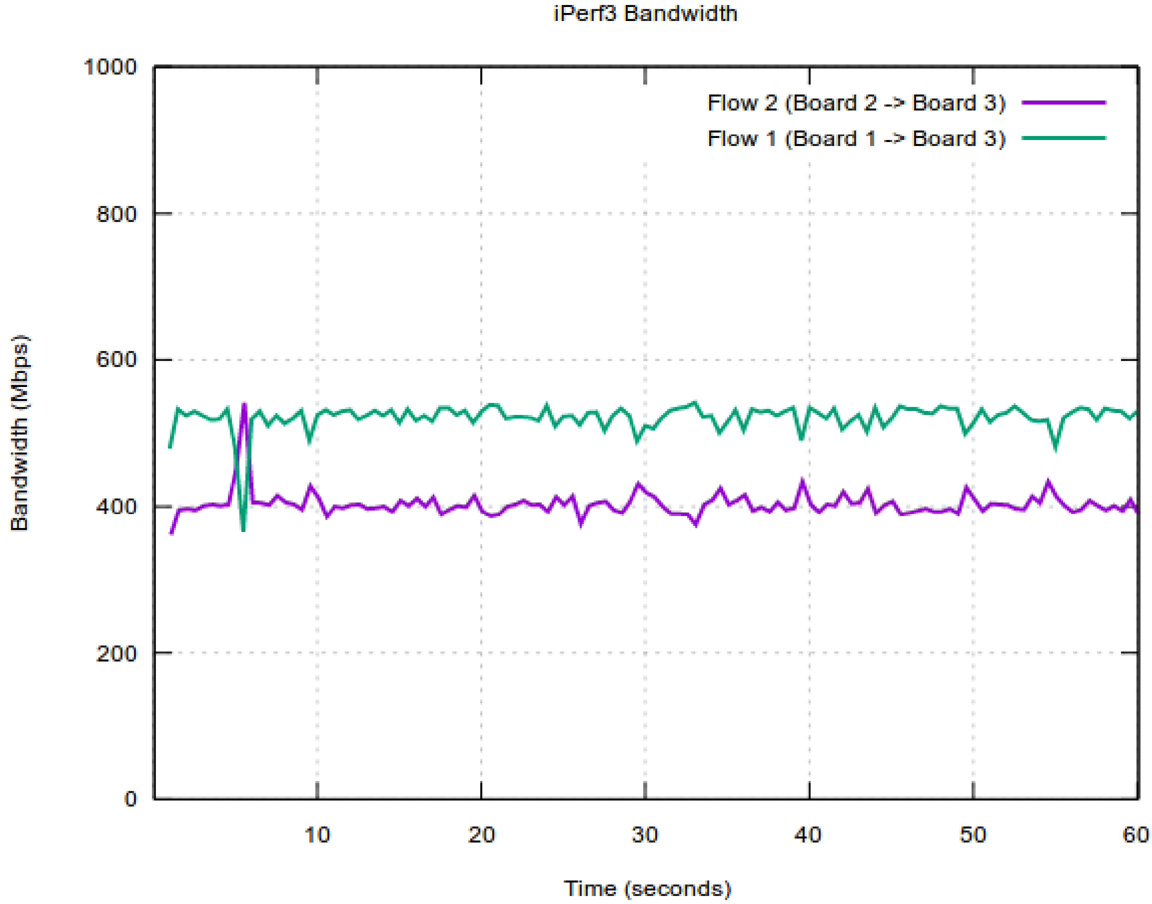


Figure 61. Policing configuration: both flows running at the same time

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.

8.2.8.8 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*.

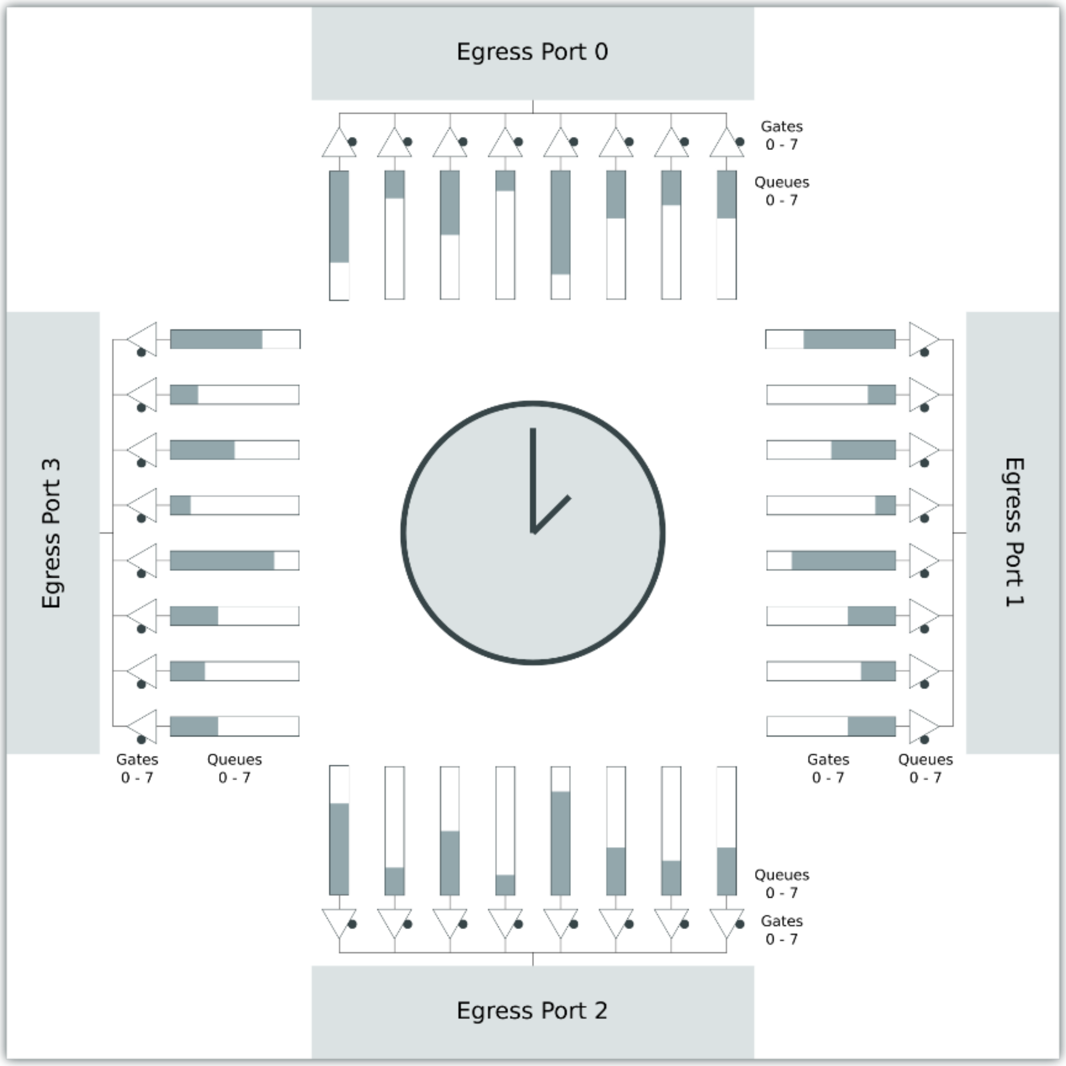


Figure 62. Structure of the Time Aware Scheduler

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.

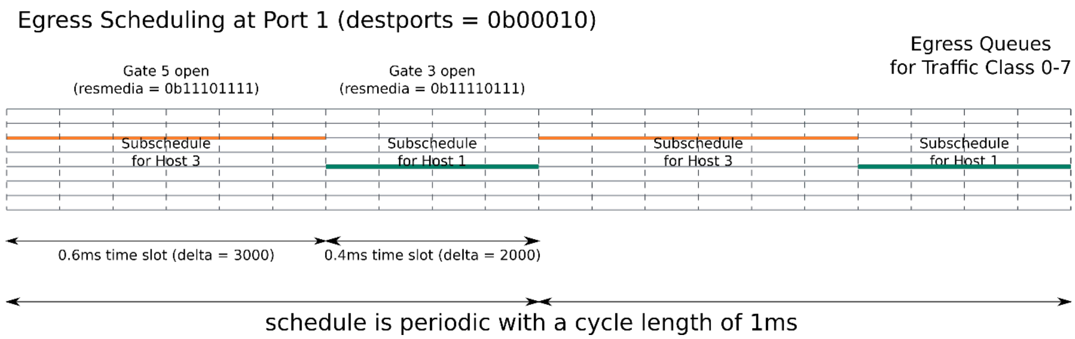


Figure 63. Time gating for Host 1 and Host 3, viewed on the time axis

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`.

8.2.8.8.1 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

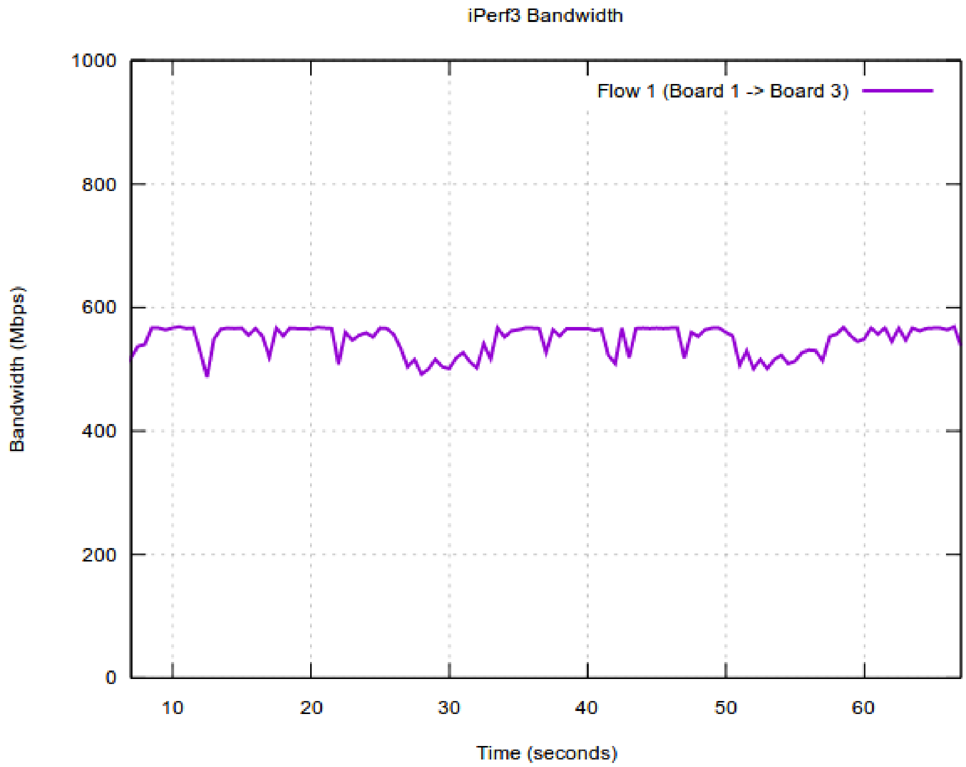


Figure 64. Scheduling configuration: Flow 1 run by its own

- Flow 2 run by its own

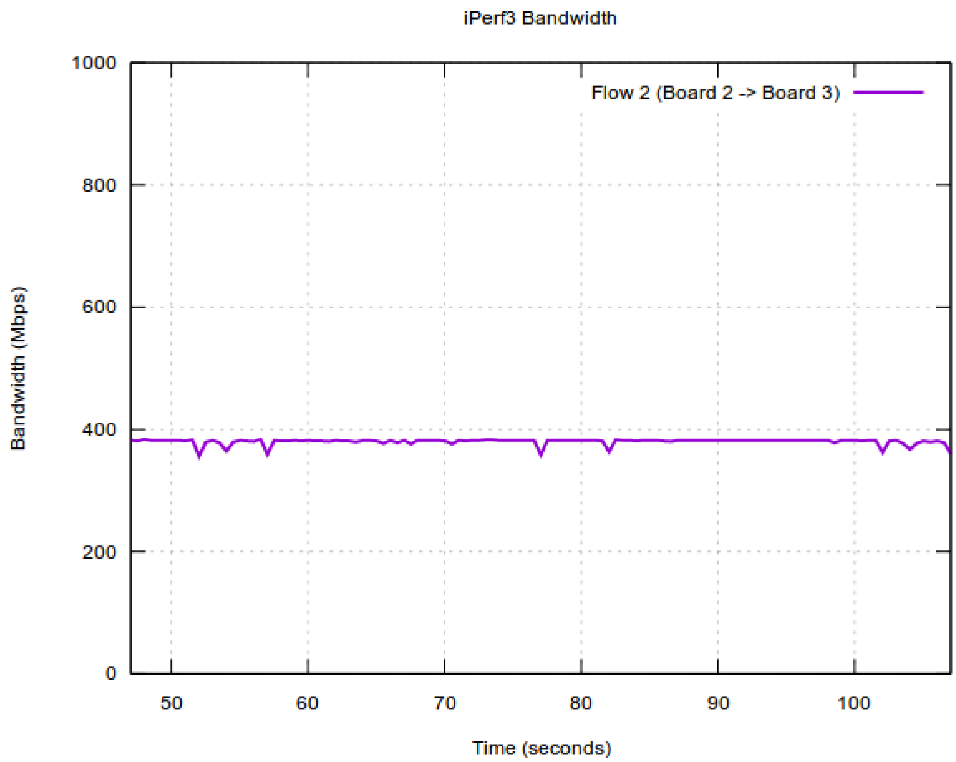


Figure 65. Scheduling configuration: Flow 2 run by its own

- Both flows run at the same time

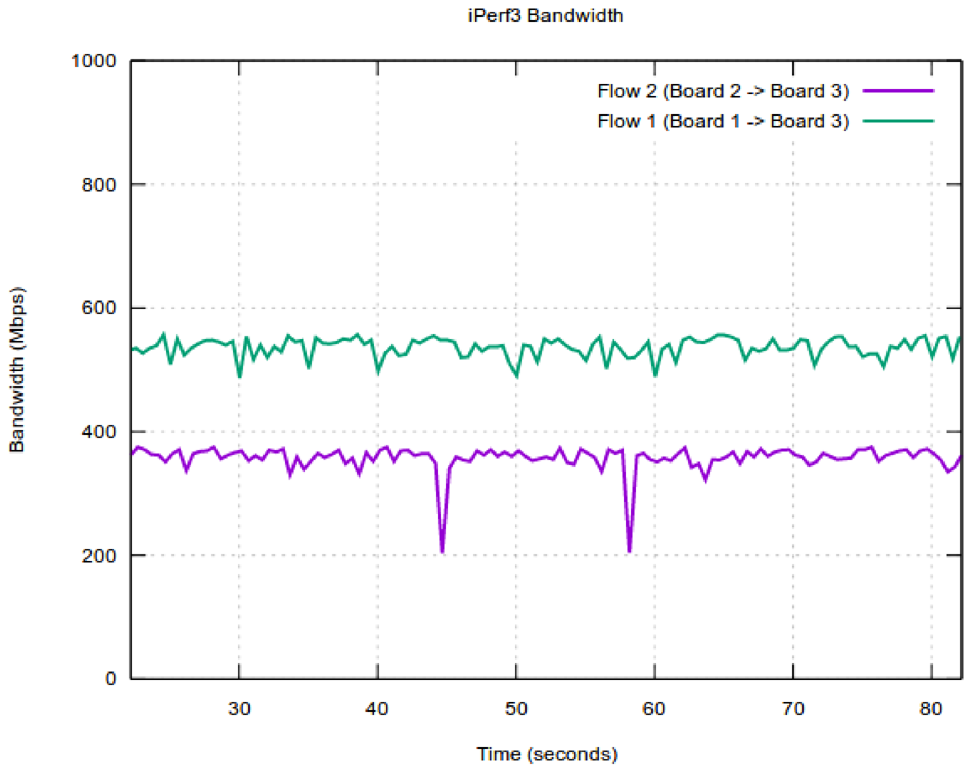


Figure 66. Both flows run at the same time

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.

8.2.8.9 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
```

8.2.9 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.

8.2.9.1 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.

8.2.9.2 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.

8.2.9.3 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:

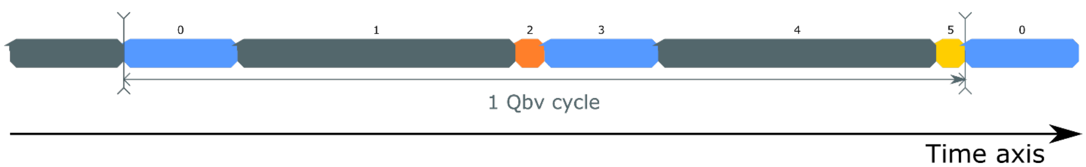
```
{
    "clksrc": "ptp",
    "cycles": [
        {
            "start-time-ms": "1",
            "timeslots": [
                {
                    "duration-ms": "4",
                    "ports": [${echo_port}, ${reply_port}],
                    "gates-open": [0, 1, 2, 3, 4, 5, 6],
    "comment": "regular traffic 1"
                },
                {
                    "duration-ms": "10",
                    "ports": [${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 67. 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.

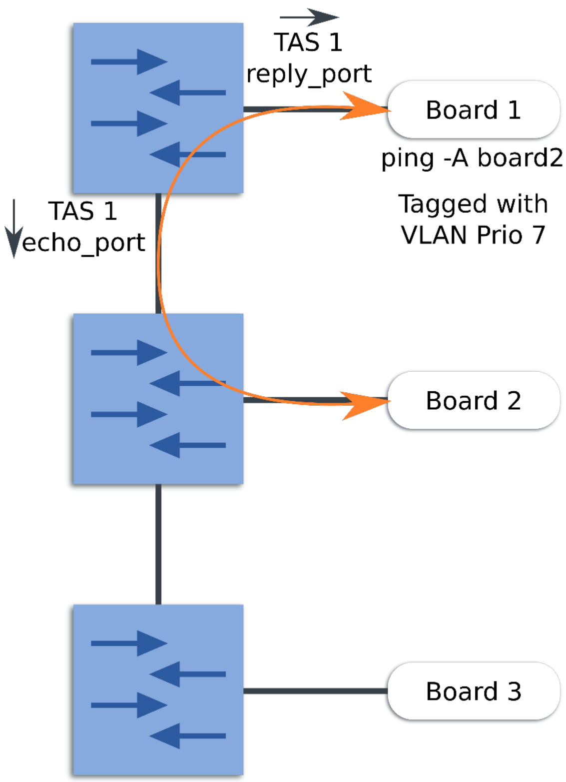
- Let the LS1021 cores of the iPerf receiver cool down from receiving Rx interrupts from the network driver and finish processing the incoming traffic. This is important because, if upon receiving an ICMP echo request during time slot 2, the destination cannot process it and generates a reply until time 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).

8.2.9.4 Scenarios

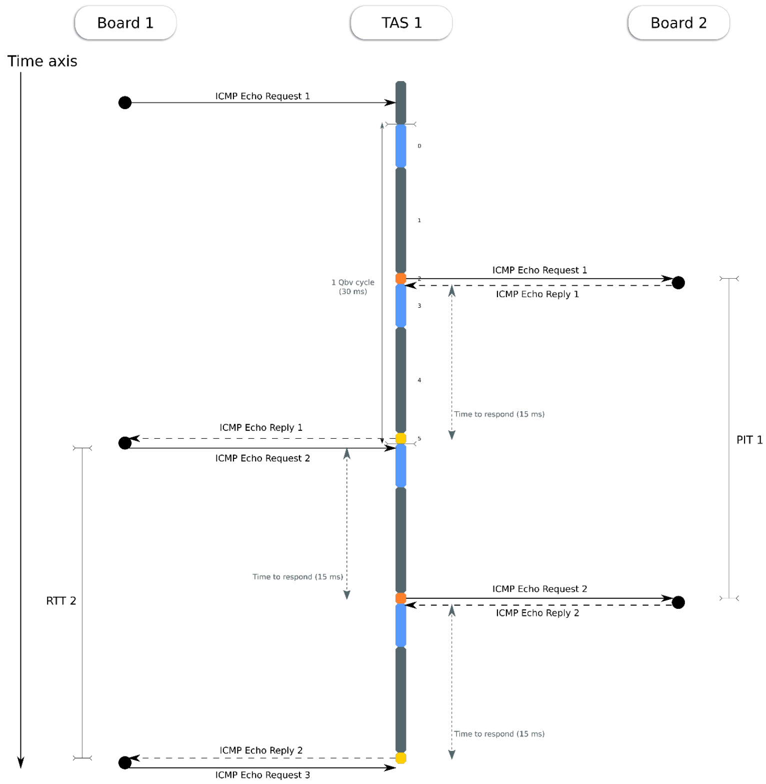
In both the 1-hop and 3-hop scenarios, adaptive ping is used to simulate the control flow packets sent from Node A to Node B. "Adaptive" ("-A" flag) means that the sending interval of Node A adapts to the RTT of the TSN network, which we are controlling directly by means of the Time-Aware Schedulers of the SJA1105 TSN switches along the path.

8.2.9.4.1 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.

8.2.9.4.2 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.

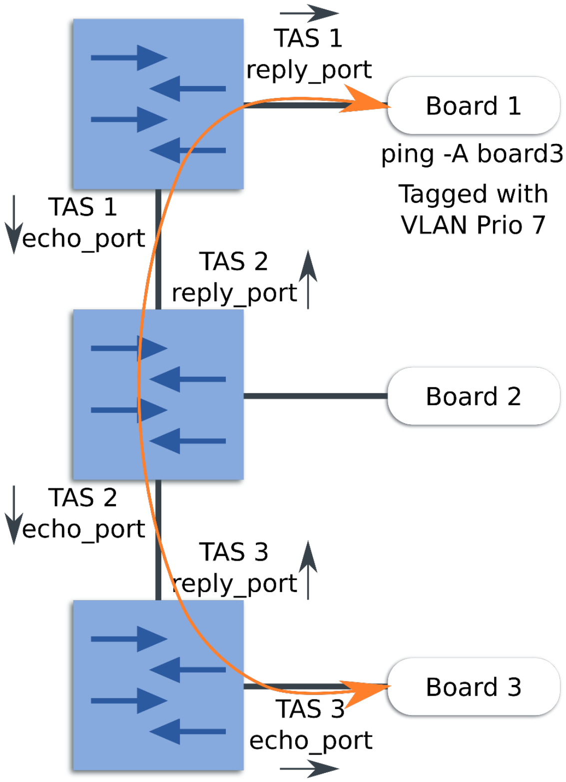


Figure 68. 3-hop scenario

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.

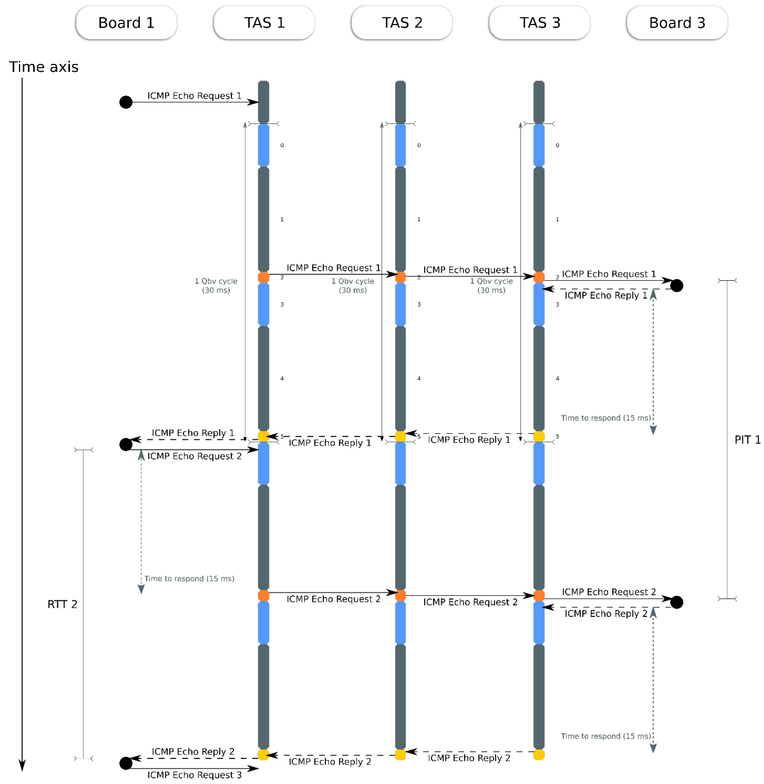


Figure 69. Timeslots in the 3 hop scenario

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.

8.2.9.5 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
```

8.2.9.6 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)

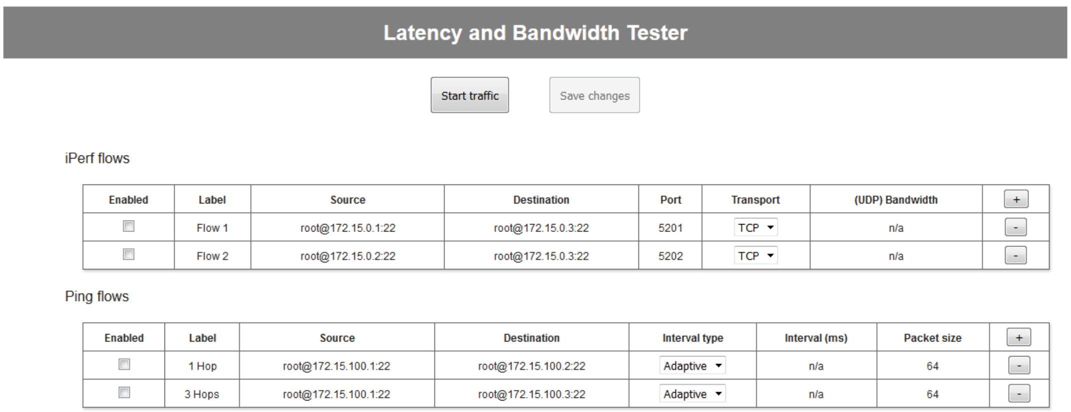


Figure 70. LBT configuration of flows for the Synchronized Qbv demo

8.2.9.7 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.

8.2.9.7.1 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.

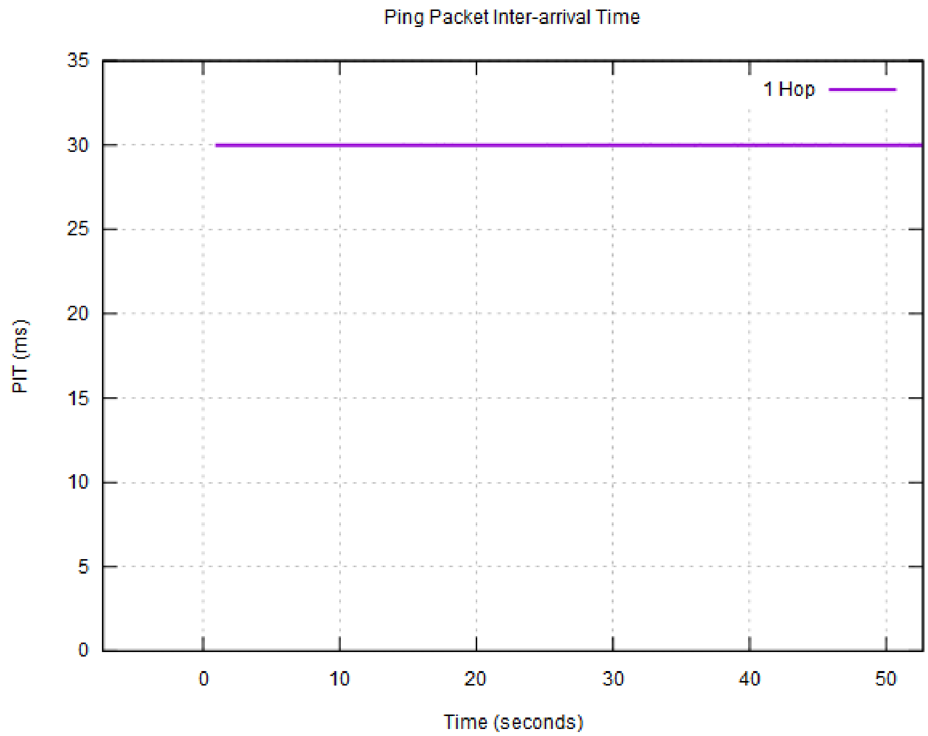


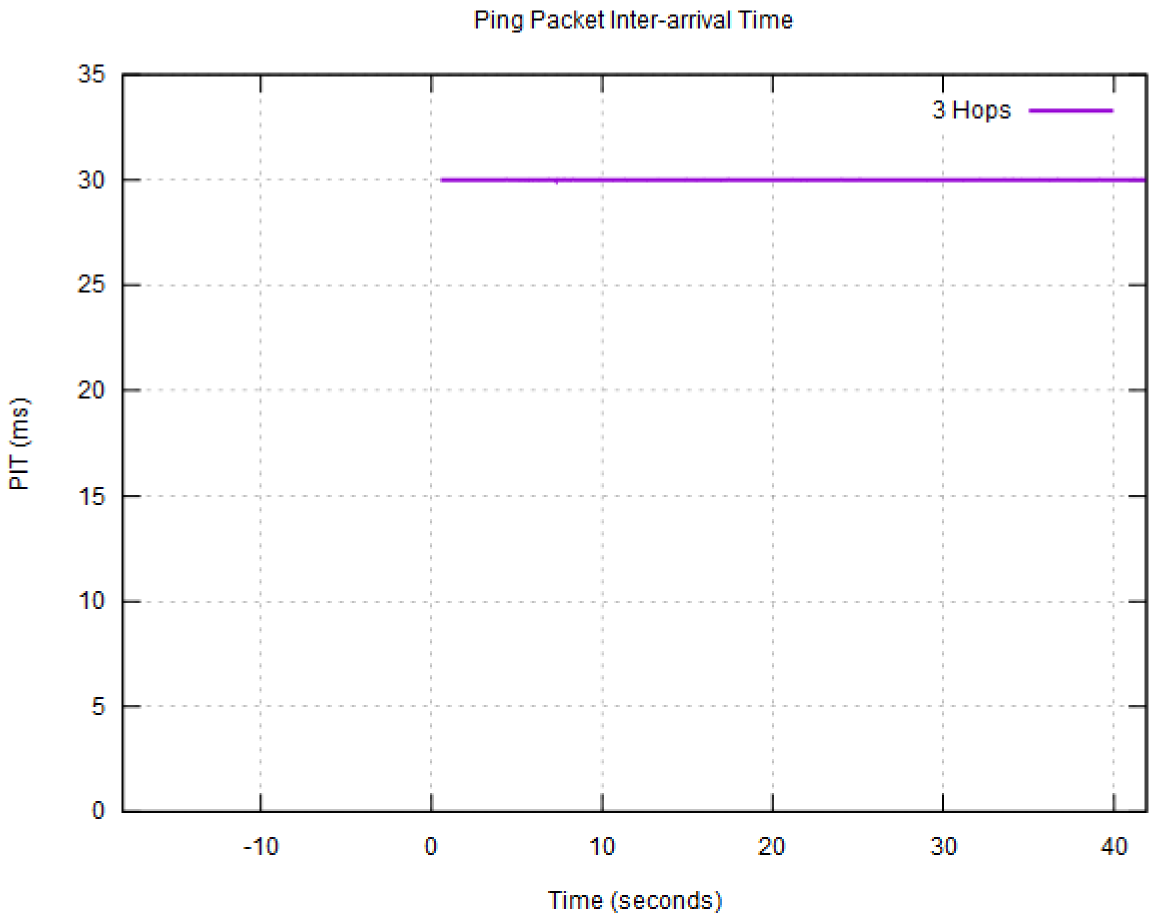
Figure 71. 1-hop flow

8.2.9.7.2 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 72. 3-hops flow



8.2.9.7.3 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.

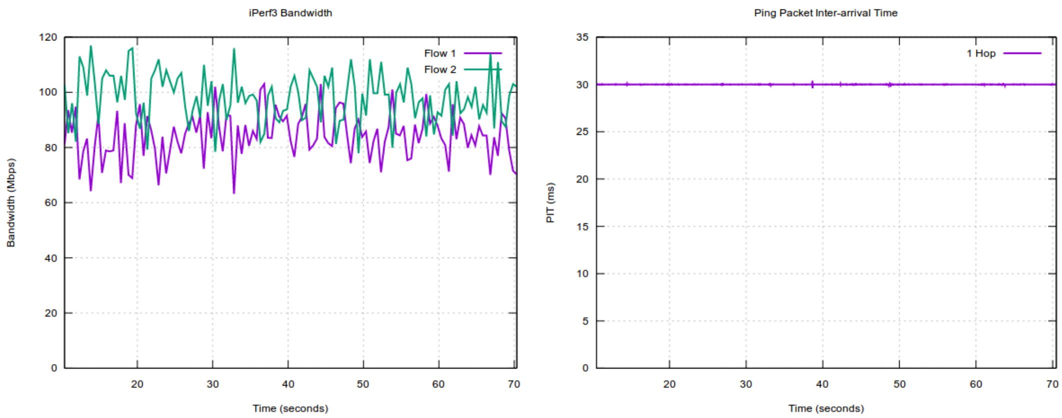


Figure 73. 1-hop ping with background traffic

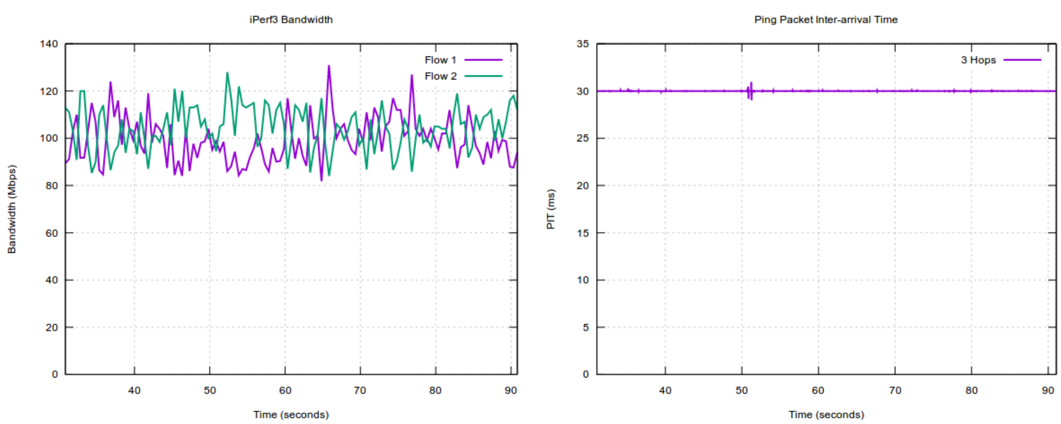
8.2.9.7.4 3 Hops with background traffic

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 74. 3-hop ping with background traffic



8.2.10 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.

8.2.10.1 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'.

For the YANG model of sja1105, click here (https://github.com/openil/openil/blob/master/package/yang-sja1105/sja1105/sja1105.yang).

8.2.10.2 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
```

8.2.10.3 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
```

8.2.10.4 Running a configuration at startup

Repeat the following commands for Board1, Board2, and Board3:

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

8.2.10.5 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.

8.2.10.6 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`

8.2.10.7 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>
```

8.2.10.8 Ending the NETCONF session

Use the following command to end the NETCONF session:

```
netconf> disconnect
```

Chapter 9
4G-LTE Modem

9.1 Introduction

4G-LTE USB modem functionality is supported on NXP's LS1021-IoT, LS1012ARDB, LS1043ARDB, LS1046ARDB, and LS1028ARDB platforms.

9.2 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).

9.3 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 --->
          <*> Multi-purpose USB Networking Framework
           <*> CDC Ethernet support
           <*> CDC EEM support
           <*> CDC NCM support
```

Finally, update the images, refer to Updating target images for LS1012ARDB.

9.4 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”.

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 10
OTA implementation

NXP's LS1021-IoT, LS1012ARDB, LS1043ARDB, LS1046ARDB, and LS1028ARDB 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.

10.1 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.

10.2 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

10.3 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.

10.4 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 11 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, LS1046ARDB, and LS1028ARDB platforms.

11.1 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.

11.2 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.

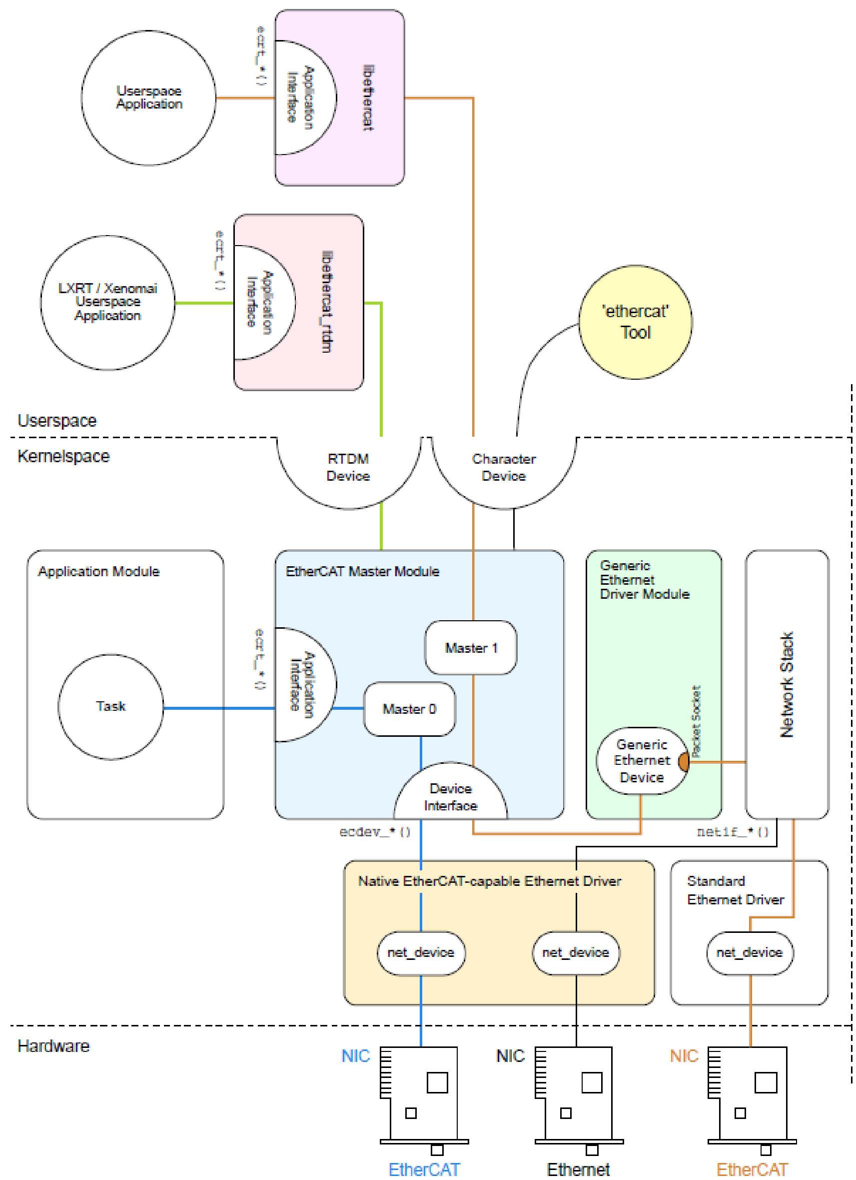


Figure 75. IGH EtherCAT master architecture

11.3 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.

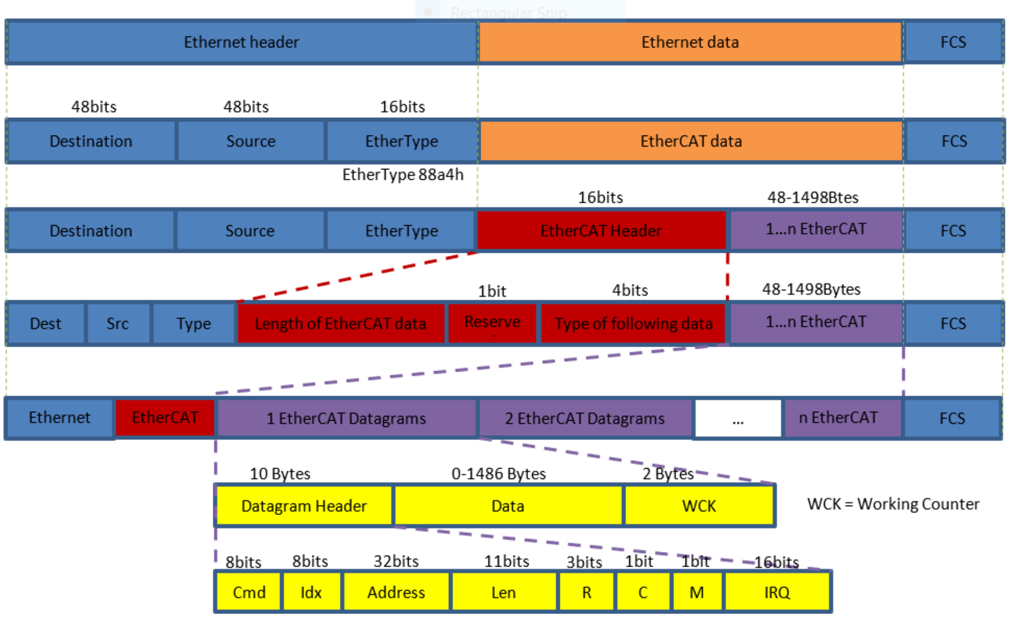


Figure 76. EtherCAT frame structure

11.4 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.

11.4.1 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.

11.4.2 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 43. 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.

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`

Table continues on the next page...

Table 43. Command line tools for EtherCAT (continued)

			`[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`

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**.

11.4.3 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
```

11.4.4 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.

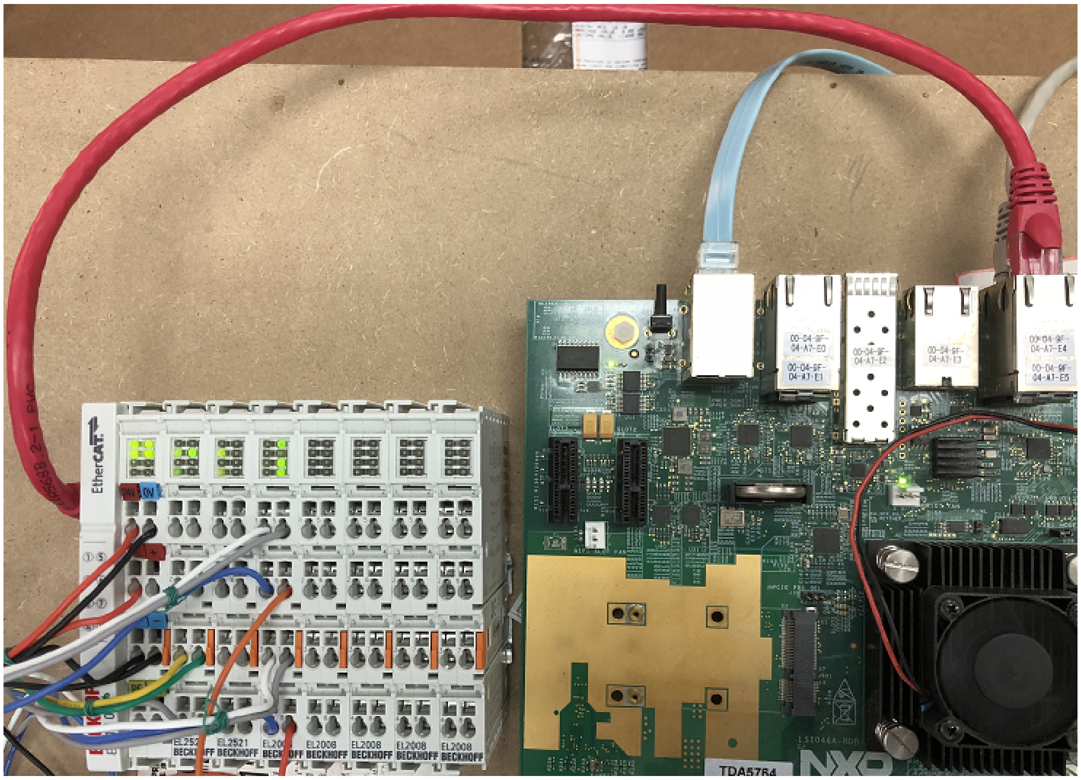


Figure 77. Board connects with BECKHOFF

- The figure below shows the BECKHOFF's EK1100 and EL2008.

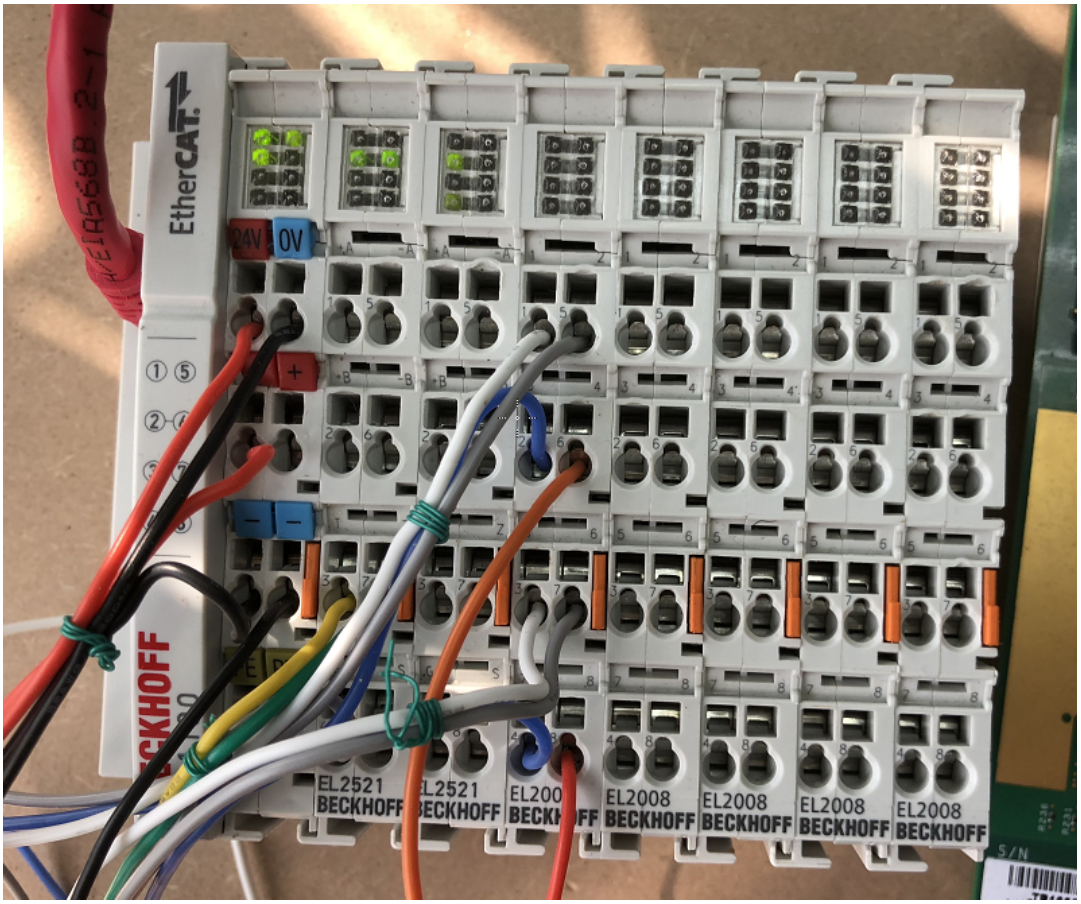


Figure 78. BECKHOFF EL2008

- The figure below shows a stepper motor driver.

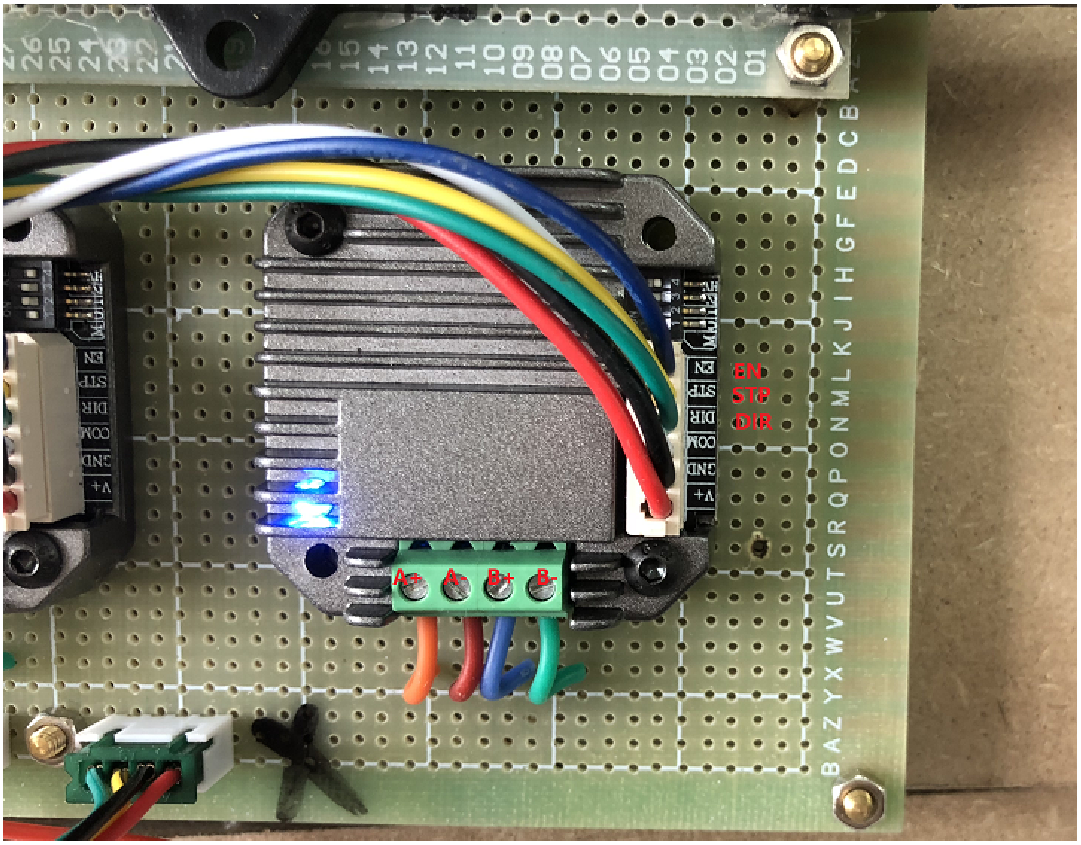


Figure 79. Stepper motor driver

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.

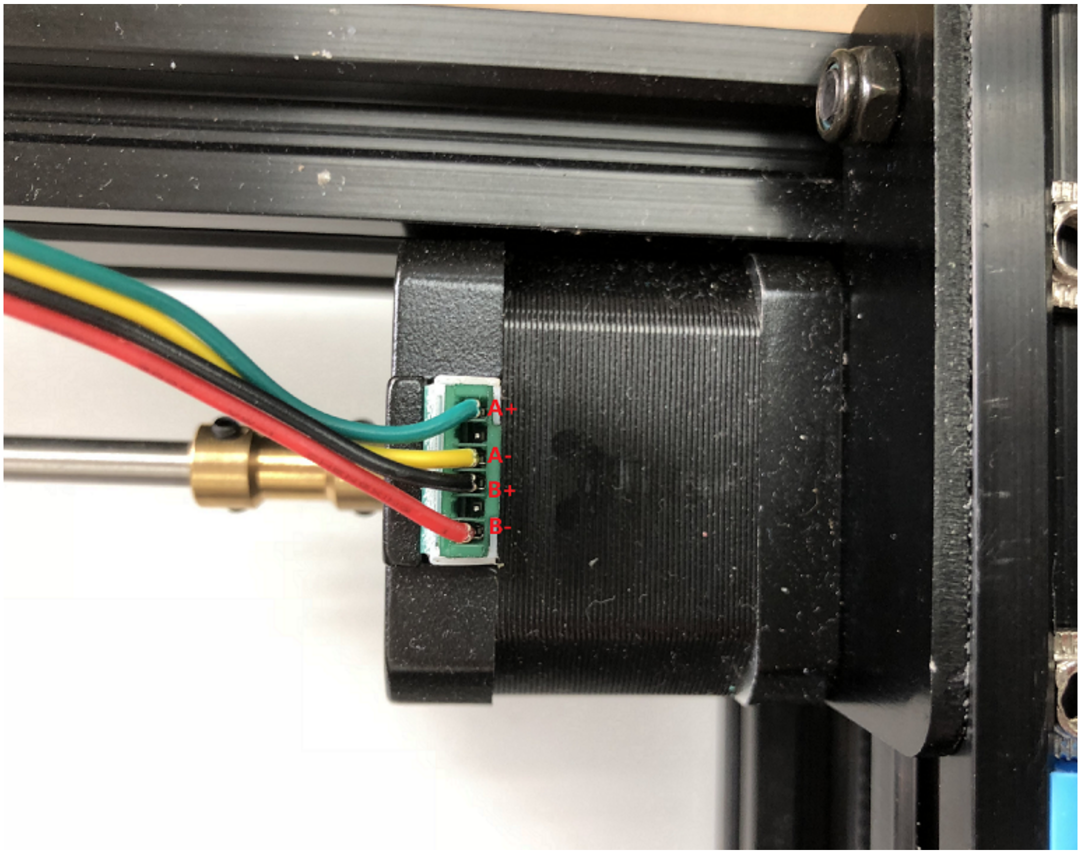


Figure 80. Stepper motor

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 12
nxp-servo

nxp-servo is a CiA402 (also referred to as DS402) profile framework based on Igh CoE interface (An EtherCAT Master stack, see EtherCAT section for details). It abstracts the CiA 402 profile and provides an easily-usable API for the Application developer.

The nxp-servo project consists of a basic library *libnservo* and several auxiliary tools.

The application developed with *libnservo* is flexible enough to adapt to the changing of CoE network by modifying the *xml* config file, which is loaded when the application starts. The *xml* config file describes the necessary information, including EtherCAT network topology, slaves configurations, masters configurations and all axles definitions.

12.1 CoE network

A typical CoE network is shown in the figure below:

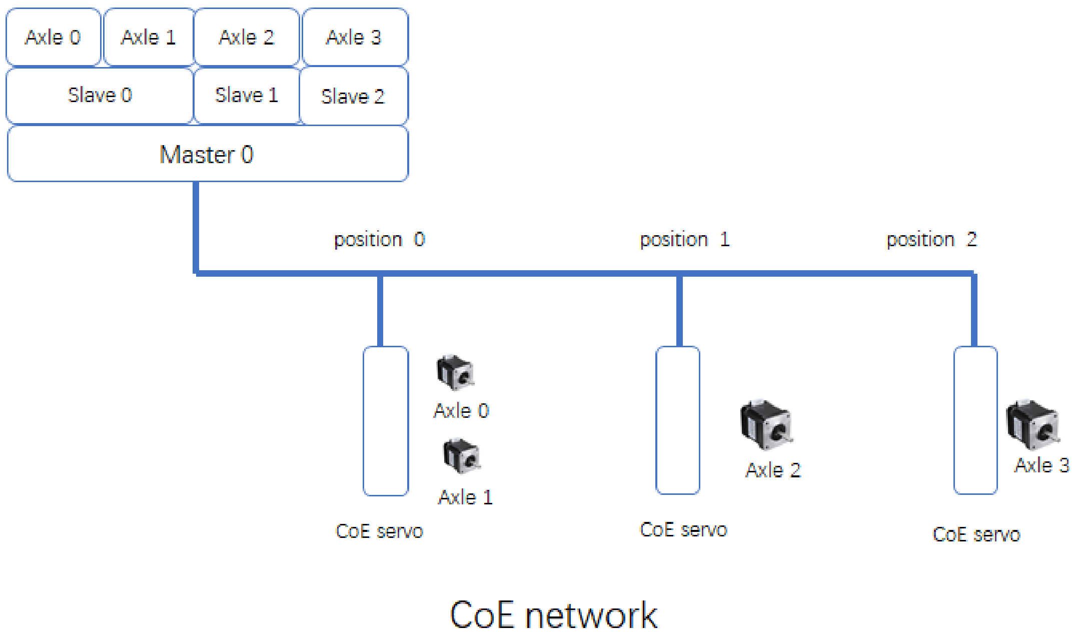


Figure 81. CoE network

There are three CoE servos on this network and we name them slave *x* as the position they are. Each CoE servo could have more then one axle. The libnservo then initiates the CoE network and encapsulates the detail of network topology into axle nodes. So the developer could focus on the each axle operation without taking care of the network topology.

12.2 Libnservo Architecture

nxp-servo is running on top of *Igh* EtherCAT stack. And the *Igh* stack provides CoE communication mechanisms - Mailbox and Process Data. Using these mechanisms, nxp-servo could access the CiA Object Dictionary located on CoE servo.

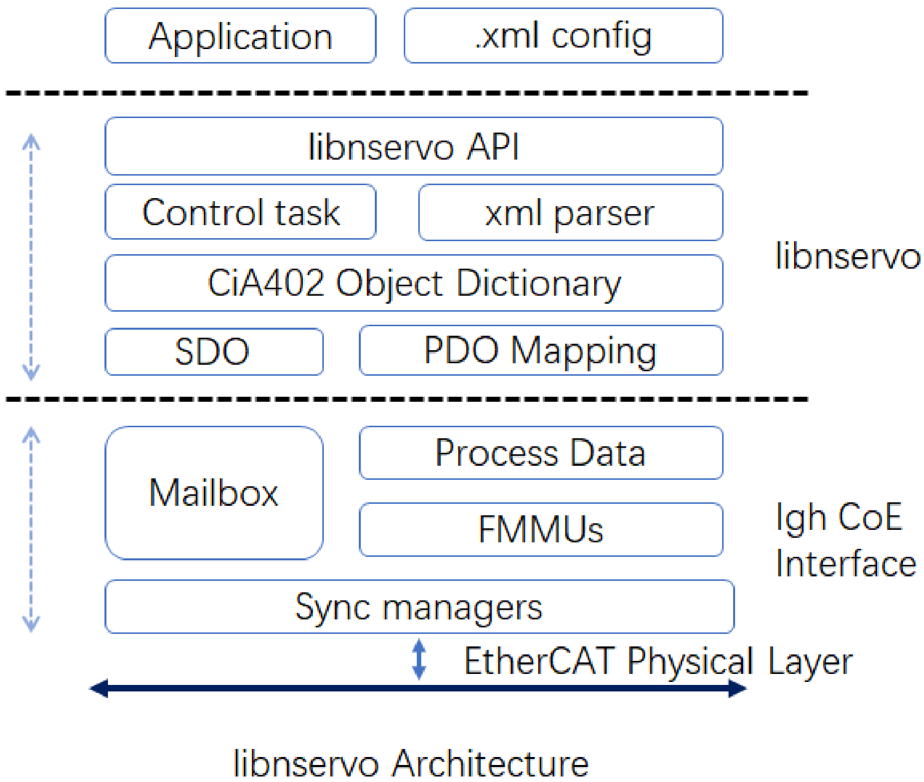


Figure 82. Libnservo architecture

Control task initiates the master, all slaves on the CoE network and registers all PDOs to Igh stack, then constructs a data structure to describe each axle. Finally, the control task creates a task to run the user task periodically.

12.3 Xml Configuration

This section focuses on how the xml config file describes a CoE network.

The skeleton of XML config is shown as in figure below:

```
<?xml version="1.0" encoding="utf-8"?>
<Config  Version="1.2">
    <PeriodTime>#10000000</PeriodTime>
    <MaxSafeStack>#8192</MaxSafeStack>
    <master_status_update_freq>#1</master_status_update_freq>
    <slave_status_update_freq>#1</slave_status_update_freq>
    <axle_status_update_freq>#1</axle_status_update_freq>
    <sync_ref_update_freq>#2</sync_ref_update_freq>
    <is_xenomai>#1</is_xenomai>
    <sched_priority>#82</sched_priority>
    <Masters>
        <Master>
            ...
        <\Master>
        <Master>
            ...
        <\Master>
    <\Master>

    <Axles>
```

```
        <Axle>
            ...
        <\Axle>
        <Axle>
            ...
        <\Axle>
    <\Axles>
</Config>
```

- All config elements must be inside the <Config> element.
- All config elements shown above are mandatory.
- The numerical value started with # means that it is a decimal value.
- The numerical value started with#x means that it is a hexadecimal value.
- <PeriodTime> element means that the period of control task is 10ms.
- <MaxSafeStack> means the stack size, and it is an estimated value. 8K is enough to satisfy most application.
- <master_status_update_freq> element means the frequency of masters status update. the value #x means update the masters status every task period.
- <slave_status_update_freq> element means the frequency of slaves status update. the value #1 means update the slaves status every task period.
- <axle_status_update_freq> element means the frequency of axles status update. the value #1 means update the axles status every task period.
- <sync_ref_update_freq> element means the frequency of reference clock update. the value #2 means update the axles status every two task period.
- <is_xenomai> element means whether Xenomai is supported. the value #1 means that Xenomai is supported on this host, and #0 means not.
- <sched_priority> element means the priority of the user task.
- <Masters> element could contain more then one Master element . For most cases, there is only one master on a host.
- <Axles> element could contain more then one Axle element, which is the developer really care about.

12.3.1 Master Element

As *CoE network* section shown, the Master could has many slaves, so the Master element will consist of some *Slave* elements.

```
    <Master>
        <Master_index>#0</Master_index>
        <Reference_clock>#0</Reference_clock>
        <Slave  alias="#0" slave_position="#0">
                ....
            </Slave>
            <Slave  alias="#1" slave_position="#1">
                ....
            </Slave>
     </Master>
```

- <Master_index> element means the index of the master. as mentioned above, for many cases, there is only one master, so the value of this element is always #0.
- <Reference_clock> element is used to indicate which slave will be used the reference clock.
- <Slave> element means there is a slave on this master.

12.3.1.1 Slave Element

```
<Slave  alias="#0" slave_position="#0">
<VendorId>#x66668888</VendorId>
<ProductCode>#x20181302</ProductCode>
    <Name>2HSS458-EC</Name>
    <Emerg_size>#x08</Emerg_size>
<WatchDog>
    <Divider>#x0</Divider>
    <Intervals>#4000</Intervals>
</WatchDog>
<DC>
    <SYNC SubIndex='#0'>
        <Shift>#0</Shift>
    </SYNC>
</DC>
<SyncManagers force_pdo_assign="#1">
    <SyncManager SubIndex="#0">
            ...
        </SyncManager>
    <SyncManager SubIndex="#1">
            ...
    </SyncManager>
</SyncManagers>
<Sdos>
    <Sdo>
        ...
    </Sdo>
    <Sdo>
    ...
    </Sdo>
</Sdos>
</Slave>
```

- *alias* attribute means the alias name of this slave.
- *slave_position* attribute means which position of the slave is on this network.
- <Name>element is the name of the slave.
- <Emerg_size> element is always 8 for all CoE device.
- <WatchDog> element is used to set the watch dog of this slave.
- <DC> element is used to set the sync info.
- <SyncManagers> element should contain all syncManager channels.
- <Sdos> element contains the default value we want to initiate by SDO channel.

12.3.1.1.1 SyncManagers Element

For a CoE device, there are generally four syncManager channels.

- SM0: Mailbox output
- SM1: Mailbox input
- SM2: Process data outputs
- SM3: process data inputs

```
<SyncManager SubIndex="#2">
    <Index>#x1c12</Index>
```

```
        <Name>Sync Manager 2</Name>
        <Dir>OUTPUT</Dir>
        <Watchdog>ENABLE</Watchdog>
        <PdoNum>#1</PdoNum>
        <Pdo SubIndex="#1">
            <Index>#x1600</Index>
            <Name>RxPdo 1</Name>
            <Entry SubIndex="#1">
                ...
            </Entry>
            <Entry SubIndex="#2">
                ...
            </Entry>
        </Pdo>
    </SyncManager>
```

- <Index> element is the object address.
- <Name> is a name of this syncmanager channel.
- <Dir> element is the direction of this syncmanager channel.
- <Watchdog> is used to set watchdog of this syncmanager channel.
- <PdoNum> element means how many PDO we want to set.
- <Pdo SubIndex="#1> element contains the object dictionary entry we want to mapped.
    - <Index> PDO address.
    - <Name> PDO name
    - <Entry> the object dictionary we want to mapped.

The Entry element is used to describe a object dictionary we want to mapped.

```
    <Entry SubIndex="#1">
        <Index>#x6041</Index>
        <SubIndex>#x0</SubIndex>
        <DataType>UINT</DataType>
        <BitLen>#16</BitLen>
        <Name>statusword</Name>
    </Entry>
```

12.3.1.1.2 Sdo Element

The Sdo element is used to set the default value of a object dictionary.

```
    <Sdo>
        <Index>#x6085</Index>
        <Subindex>#x0</Subindex>
        <value>#x1000</value>
        <BitLen>#32</BitLen>
        <DataType>DINT</DataType>
        <Name>Quick_stop_deceleration</Name>
    </Sdo>
```

The element shown in figure above means set the Object Dictionary "6085" to 0x1000.

12.3.2 Axle Element

```
<Axle master_index='#0' slave_position="#0" AxleIndex="#0" AxleOffset="#0">
    <Mode>pp</Mode>
    <Name>x-axle</Name>
    <reg_pdo>
    ...
    </reg_pdo>
    <reg_pdo>
    ...
    </reg_pdo>
</Axle>
```

- *master_index* attribute indicates which *master* this *axle* belong to.
- *slave_position* attribute indicates which *slave* this *axle* belong to.
- *AxleOffset* attribute indicates which *axle* this *axle* is on the slave. As mentioned above, a CoE slave could have more then on *axle* . If this axle is the second axle on the slave, set *AxleOffset="#1"* .
- <Mode> means which mode this axle will work on.
- <Name> is the name of this axle.
- <reg_pdo> is the PDO entry we want to register.

reg_pdo element

```
<reg_pdo>
    <Index>#x606c</Index>
    <Subindex>#x0</Subindex>
    <Name></Name>
</reg_pdo>
```

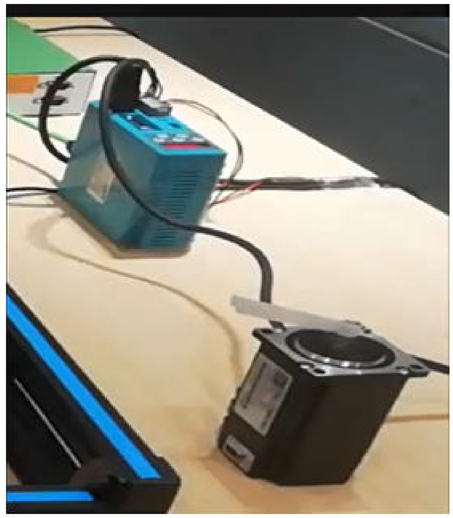
12.4 Test

12.4.1 Hardware Preparation

- A CoE servo system

  A CoE servo system includes a CoE servo and a motor. In this test, '2HSS458-EC' servo system shown as in figure below will be used.
- A board supported on OpenIL

  In this test, LS1046ARDB will be used.



2HSS458-EC Servo System

12.4.2 Software Preparation

Make sure the below config options is selected when configuring OpenIL.

- BR2_PACKAGE_IGH_ETHERCAT=y
- BR2_PACKAGE_LIBXML2=y
- BR2_PACKAGE_NXP_SERVO=y

12.4.3 CoE Network Detection

- Igh configuration
  - Configure the MASTER0_DEVICE field of the */etc/ethercat.conf*

    Set MASTER0_DEVICE to the MAC address to indicate which port the Igh uses .
  - Configure DEVICE_MODULES="generic" of the */etc/ethercat.conf*
- Using the command

  ```
  [root@OpenIL:~]#ethercatctl start
  ```

  to start Igh service.
- Check CoE servo using below command.

  ```
  [root@OpenIL:~]#ethercat slaves
  0  0:0  PREOP  +  2HSS458-EC
  ```

12.4.4 Start Test

Note: The *Position encoder resolution* and *Velocity encoder resolution* of "2HSS458-EC" servo system are both 4000 . It means the ratio of encoder increments per motor revolution.

- *Profile Position* mode test

— Start the test service as below.

```
[root@OpenIL:~]# nservo_run -f /root/nservo_example/hss248_ec_config_pp.xml &
```

— Check whether the status of the slave has been transferred from "PREOP" to "OP".

```
[root@OpenIL:~]# ethercat slaves
0  0:0  OP  +  2HSS458-EC
```

— Check whether the phase of the master has been transferred from "Idle" to "Operation".

```
[root@OpenIL:~]# ethercat master | grep Phase
  Phase: Operation
```

— Run below commands to test whether the motor works.

- Get current mode of axle 0.

```
[root@OpenIL:~]# nservo_client -a 0 -c get_mode
get_mode of the axle 0 : Profile Position Mode
```

- Get current position of axle 0.

```
[root@OpenIL:~]# nservo_client -a 0 -c get_position
get_current_position of the axle 0 : 0
```

- Get the profile speed of axle 0.

```
[root@OpenIL:~]# nservo_client -a 0  -c get_profile_speed
get_profile_speed of the axle 0 : 800000
```

The value 800000 means 200 revolutions per second.

- Set profile speed of axle 0.

```
[root@OpenIL:~]# nservo_client -a 0  -c set_profile_speed:20000
set_profile_speed of the axle 0 : 20000
```

Set profile speed to 5 revolutions per second.

- Set target position of axle 0

```
[root@OpenIL:~]# nservo_client -c set_position:400000
set_position of the axle 0 : 400000
```

The value 400000 means that the motor will turn 100 rounds.

(target_position:400000 - current_position:0) / 4000 = 100

- Get current speed of axle 0

```
[root@OpenIL:~]# nservo_client -a 0  -c get_speed
get_speed of the axle 0 : 19999
```

- Get target position of axle 0

```
[root@OpenIL:~]# nservo_client -a 0  -c get_target_position
get_target_position of the axle 0 : 400000
```

— Exit

```
[root@OpenIL:~]# nservo_client  -c exit
```

- *Profile Velocity* mode test
  - — Start the test service as below.

```
[root@OpenIL:~]# nservo_run -f /root/nservo_example/hss248_ec_config_pv.xml &
```

  - — Check whether the status of the slave has been transferred from "PREOP" to "OP".

```
[root@OpenIL:~]# ethercat slaves
0  0:0  OP  +  2HSS458-EC
```

  - — Check whether the phase of the master has been transferred from "Idle" to "Operation".

```
[root@OpenIL:~]# ethercat master | grep Phase
  Phase: Operation
```

  - — Run below commands to test whether the motor works.
    - ◦ Get current mode of axle 0.

```
[root@OpenIL:~]# nservo_client -a 0  -c get_mode
get_mode of the axle 0 : Profile Velocity Mode
```

    - ◦ Set target speed of axle 0.

```
[root@OpenIL:~]# nservo_client -a 0  -c set_speed:40000
set_speed of the axle 0 : 40000
```

      The value 40000 means that the motor will turn with 10 revolutions per second.

    - ◦ Get current speed of axle 0.

```
[root@OpenIL:~]# nservo_client -a 0  -c get_speed
get_speed of the axle 0 : 32000
```

    - ◦ Get target speed of axle 0.

```
[root@OpenIL:~]# nservo_client -a 0  -c get_target_speed
get_target_speed of the axle 0 : 40000
```

  - — Exit

```
[root@OpenIL:~]# nservo_client  -c exit
```

Chapter 13 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.

13.1 Introduction

Both the LS1021A and LS1028A boards have 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.

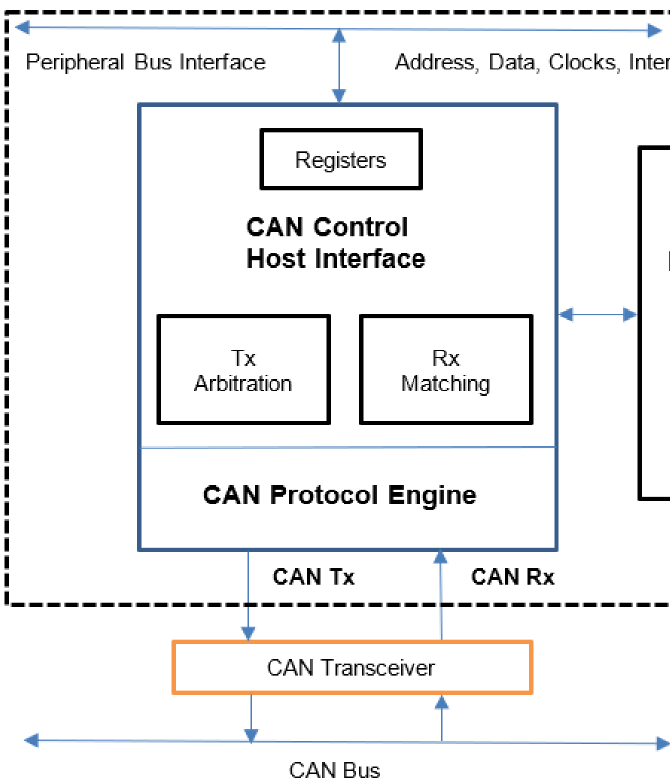


Figure 83. FlexCAN block diagram

13.1.1 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.

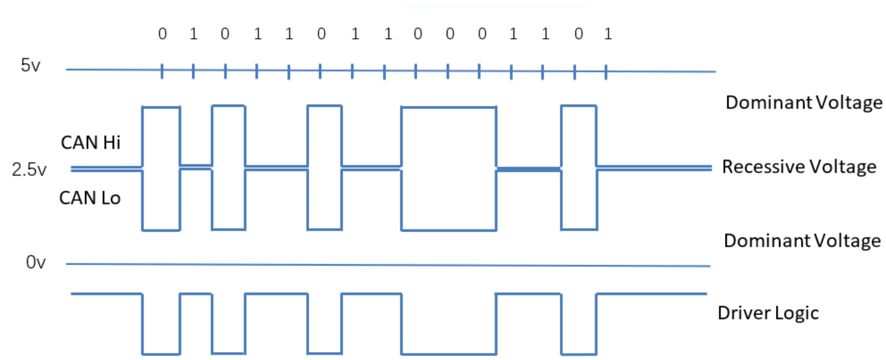


Figure 84. High speed CAN signaling

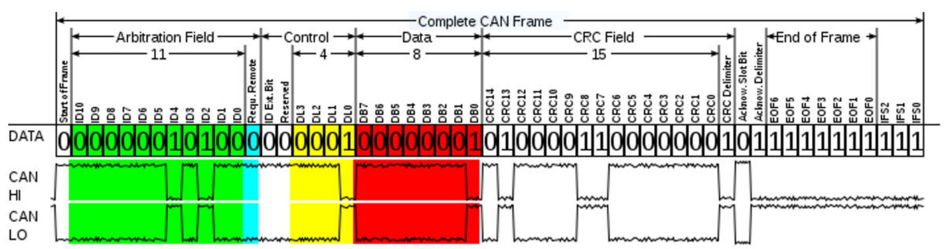


Figure 85. Base frame format

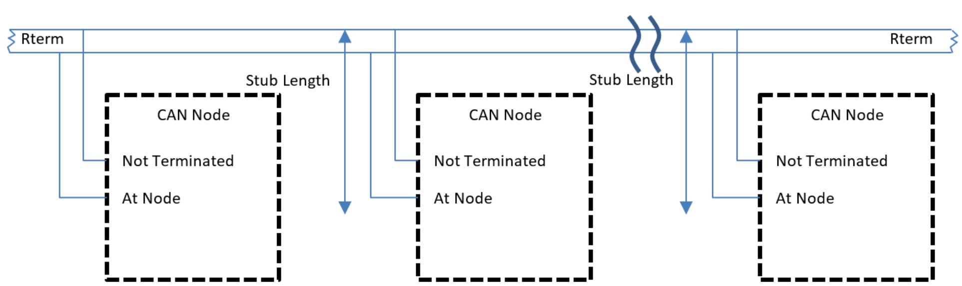


Figure 86. High speed CAN network

13.1.2 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.

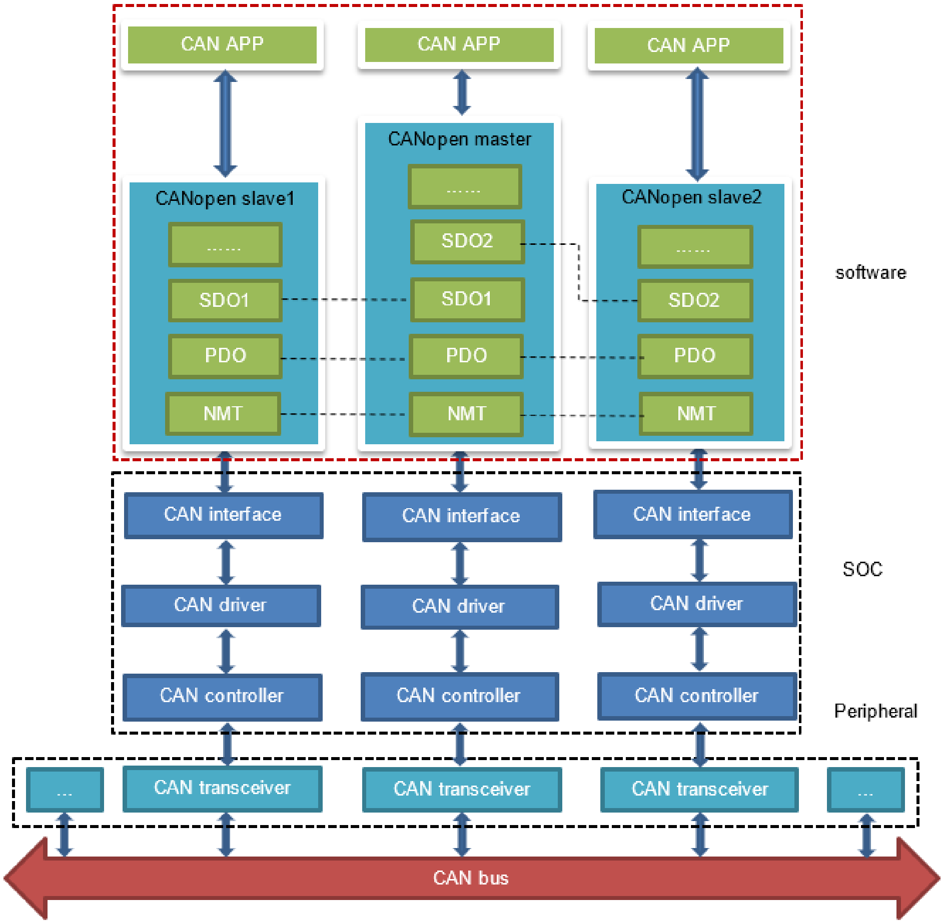


Figure 87. CANopen architecture

13.2 FlexCAN integration in OpenIL

For LS1021A, there are four CAN controllers. 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. For LS1028A, there are two CAN controllers, CAN1 and CAN2, and both of them are used in LS1028ARDB board.

13.2.1 LS1021AIOT CAN 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
```

13.2.2 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 44. 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
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

Table continues on the next page...

Table 44. CAN Net APIs and their description (continued)

API name (type)	Description
	• 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

13.3 Running a CAN application

The following sections describe the hardware and software preparation steps for running a CAN application. The hardware preparation is described separately for the LS1021A-IoT and LS1028ARDB, but the sections Compiling the CANopen-app binary for the master node, Running the CANopen application, and Running the Socketcan commands are applicable to both LS1021A-IoT and LS1028A platforms.

13.3.1 Hardware preparation for LS1021-IoT

For LS1021-IoT, 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)

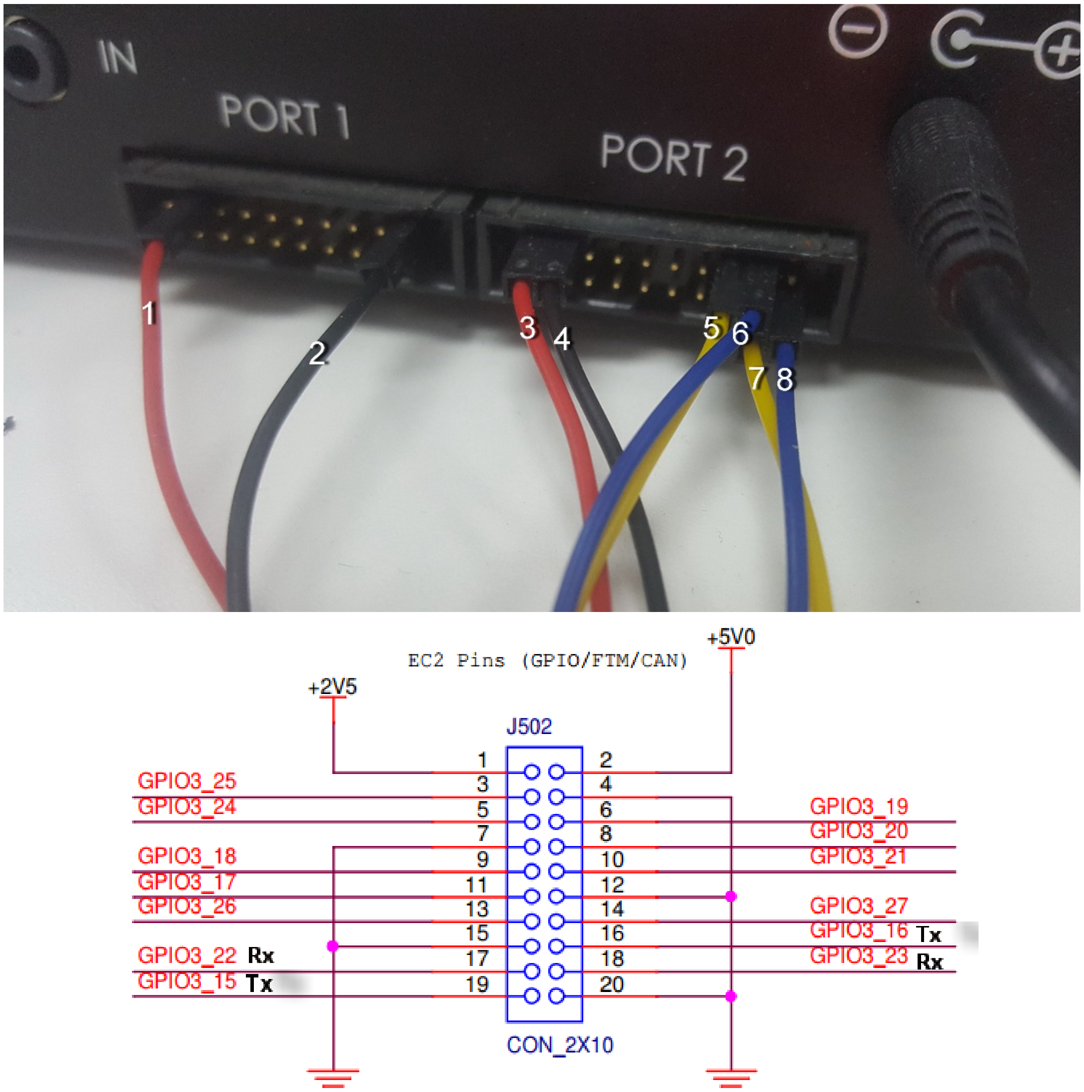


Figure 88. Hardware diagram for the FlexCan demo

NOTE

— Line1 and line3 are 5.0 V.

— Line2 and line4 are GND.

— Line5 is CAN3 Tx.

— Line6 is CAN3 Rx.

— Line7 is CAN4 Rx.

— Line8 is CAN4 Tx.

13.3.2 Hardware preparation for LS1028ARDB

For LS1028ARDB, below hardware is required:

- LS1028ARDB board
- Two cables to connect CAN1 and CAN.

The hardware connection diagram is as shown in the following figure

Figure 89. Physical connection for CAN using LS1028ARDB

13.3.3 Compiling the CANopen-app binary for the master node

This section describes the procedure for compiling the CANopen-app binary for the master node, for both LS1021A and LS1028A platforms.

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 https://github.com/openil/openil.git
$ cd openil # checkout to OpenIL-201904
$ make nxp_ls1021aiot_baremetal_defconfig
# or
$ make nxp_ls1028ardb-64b_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 to enable the below options 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.

13.3.4 Running the CANopen application

This section describes the procedure for running the CANopen-app application, which is the same for both LS1021A and LS1028A platforms.

1. First, boot the LS1021A-IoT or LS1028ARDB board.
2. Then, run the `CANopen-app` command in any directory in Linux prompt. While executing this command, first run the test code.
3. After the test code is completed, you can implement the required instructions. The command `CANopen-app` execution process steps are described below:
   a. 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!
```

b. There are nine test codes in total, tests 1 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.

c. 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
```

13.3.5 Running the Socketcan commands

This section describes the steps for running Socketcan commands that can be performed on either of the boards (LS1021A-IoT or LS1021ARDB). These commands are executed on Linux. The standard Socketcan commands are the following:

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.

13.3.6 Testing CAN bus

Below is the sample code for testing the CAN bus on LS1028ARDB.

```
[root@OpenIL:~]# ip link set can0 down
[root@OpenIL:~]# ip link set can1 down
[root@OpenIL:~]# ip link set can0 type can loopback off
[root@OpenIL:~]# ip link set can1 type can loopback off
```

```
[root@OpenIL:~]# ip link set can0 type can bitrate 500000
[root@OpenIL:~]# ip link set can1 type can bitrate 500000
[root@OpenIL:~]# ip link set can0 up
[root@OpenIL:~]# ip link set can1 up
[root@OpenIL:~]# candump can0 &
[root@OpenIL:~]# candump can1 &
[root@OpenIL:~]# cansend can0 001#224466
  can0  001   [3]  22 44 66
[root@OpenIL:~]#   can1  001   [3]  22 44 66
[root@OpenIL:~]# cansend can1 001#224466
  can0  001   [3]  22 44 66
  can1  001   [3]  22 44 66
[root@OpenIL:~]# cansend can1 001#113355
  can0  001   [3]  11 33 55
  can1  001   [3]  11 33 55
[root@OpenIL:~]# cansend can0 000#224466
  can0  000   [3]  22 44 66
```

Chapter 14
NFC click board

NFC click board is a mikroBUS™ add-on board with a versatile near field communications controller from NXP — the PN7120 IC. NFC devices are used in contactless payment systems, electronic ticketing, smartcards, but also in retail and advertising — inexpensive NFC tags can be embedded into packaging labels, flyers or posters.

This board is fully compliant with NFC Forum specifications. This implies that users can use the full potential of NFC and its three distinct operating modes listed below:

1. Card emulation
2. Read/Write
3. P2P

14.1 Introduction

The NXP's PN7120 IC integrates an ARM™ Cortex-M0 MCU, which enables easier integration into designs, because it requires fewer resources from the host MCU. The integrated firmware provides all NFC protocols for performing the contactless communication in charge of the modulation, data processing and error detection.

The board communicates with the target board MCU through the mikroBUS™ I2C interface, in compliance with NCI 1.0 host protocols (NCI stands for NFC controller interface). RST and INT pins provide additional functionality. The board uses a 3.3V power supply.

14.2 PN7120 features

PN7120 embeds a new generation RF contactless front-end supporting various transmission modes according to NFCIP-1 and NFCIP-2, ISO/IEC14443, ISO/IEC 15693, ISO/IEC 18000-3, MIFARE and FeliCa specifications. It embeds an ARM Cortex-M0 microcontroller core loaded with the integrated firmware supporting the NCI 1.0 host communication.

14.3 Hardware preparation

Use the following hardware items for the NFC clickboard demo setup:

1. LS1028ARDB
2. NFC Click board
3. NFC Sample Card (tag)

NOTE

You need to insert the NFC click board into the LS1028ardb mikroBUS1 slot.

14.4 Software preparation

In order to support NFC click board, use the following steps:

1. In OpenIL environment, use the command `make menuconfig` to enable the below options:

```
$make menuconfig
Target packages --->
         Libraries --->
                 Hardware handling --->
                        [*] libnfc-nci
```

2. In Linux kernel environment, make sure the below options are enabled:

```
$make linux-menuconfig
Device Drivers --->
    Misc devices --->
        <M> NXP PN5XX based driver
```

NOTE

The **NXP PN5XX based driver** only supports the Module mode.

3. Use the `make` command to create the images.

14.5 Testing the NFC click board

Use the following steps for testing the NFC Clickboard:

1. Install NFC driver module

```
[root@OpenIL:~]# insmod /lib/modules/4.14.47-ipipe/kernel/drivers/misc/nxp-pn5xx/pn5xx_i2c.ko
```

2. The following logs appear at the console after the above command is successful. The error information can be ignored in this case.

```
[root@OpenIL:~]# insmod /lib/modules/4.14.47-ipipe/kernel/[  195.547601] random: crng init
done
[  195.551016] random: 5 urandom warning(s) missed due to ratelimiting
[root@OpenIL:~]# insmod /lib/modules/4.14.47-ipipe/kernel/drivers/misc/nxp-pn5xx
/pn5xx_i2c.ko
[  777.503246] pn54x_dev_init
[  777.506048] pn54x_probe
[  777.508523] pn544 7-0028: FIRM GPIO <OPTIONAL> error getting from OF node
[  777.515344] pn544 7-0028: CLKREQ GPIO <OPTIONAL> error getting from OF node
[  777.522347] pn544 7-0028: 7-0028 supply nxp,pn54x-pvdd not found, using dummy regulator
[  777.530424] pn544 7-0028: 7-0028 supply nxp,pn54x-vbat not found, using dummy regulator
[  777.538490] pn544 7-0028: 7-0028 supply nxp,pn54x-pmuvcc not found, using dummy regulato
r
[  777.546723] pn544 7-0028: 7-0028 supply nxp,pn54x-sevdd not found, using dummy regulator
```

3. Run the nfcDemoApp application

```
[root@OpenIL:~]# nfcDemoApp poll
```

```
[root@OpenIL:~]# nfcDemoApp poll
##########################################################################################
##                                        NFC demo                                      ##
##########################################################################################
##                                  Poll mode activated                         [ 1251.20807
1] pn54x_dev_open : 10,55
        ##
####[ 1251.212807] pn54x_dev_ioctl, cmd=1074063617, arg=1
################[ 1251.219006] pn544_enable power on
######################################################################
                              ... press enter to quit ...

[ 1251.431597] pn54x_dev_ioctl, cmd=1074063617, arg=0
[ 1251.436416] pn544_disable power off
[ 1251.647586] pn54x_dev_ioctl, cmd=1074063617, arg=1
[ 1251.652401] pn544_enable power on
NfcHcpX:8103
NfcHcpR:8180
NfcHcpX:810103020304
NfcHcpR:8180
NfcHcpX:81010143da67663bda6766
NfcHcpR:8180
NfcHcpX:810204
NfcHcpR:818000
Waiting for a Tag/Device...
```

4. Put the NFC Sample Card (tag) on top of the NFC click board:

```
Waiting for a Tag/Device...

        NFC Tag Found

        Type :         'Type A - Mifare Ul'
        NFCID1 :        '04 67 66 D2 9C 39 81 '
                Record Found :
                                NDEF Content Max size :     '868 bytes'
                                NDEF Actual Content size :     '29 bytes'
                                ReadOnly :                      'FALSE'
                Read NDEF URL Error

                29 bytes of NDEF data received :
                D1 01 19 55 01 6E 78 70 2E 63 6F 6D 2F 64 65 6D 6F 62 6F 61 72 64 2F 4F 4D
35 35 37 38

        NFC Tag Lost

Waiting for a Tag/Device...
```

Printing the above information indicates successful card reading.

Chapter 15
BEE Click Board

This chapter introduces the features of the BEE Click Board and how to use it on LS1028ARDB.

15.1 Introduction

The BEE Click Board features the MRF24J40MA 2.4 GHz IEEE 802.15.4 radio transceiver module from Microchip. The click is designed to run on 3.3 V power supply only. It communicates with the target controller over an SPI interface.

15.2 Features

The features of the BEE Click Board are listed below:

- PCB antenna
- MRF24J40MA module
- Low current consumption (Tx 23 mA, Rx 19 mA, Sleep 2 μA)
- ZigBee stack
- MiWi™ stack
- SPI Interface
- 3.3 V power supply

15.3 Hardware preparation

Use the following hardware items for the BEE Click Board demo setup:

- Two LS1028ARDB Boards
- Two BEE Click Boards

The figure below describes the hardware setup for the BEE Click Board.

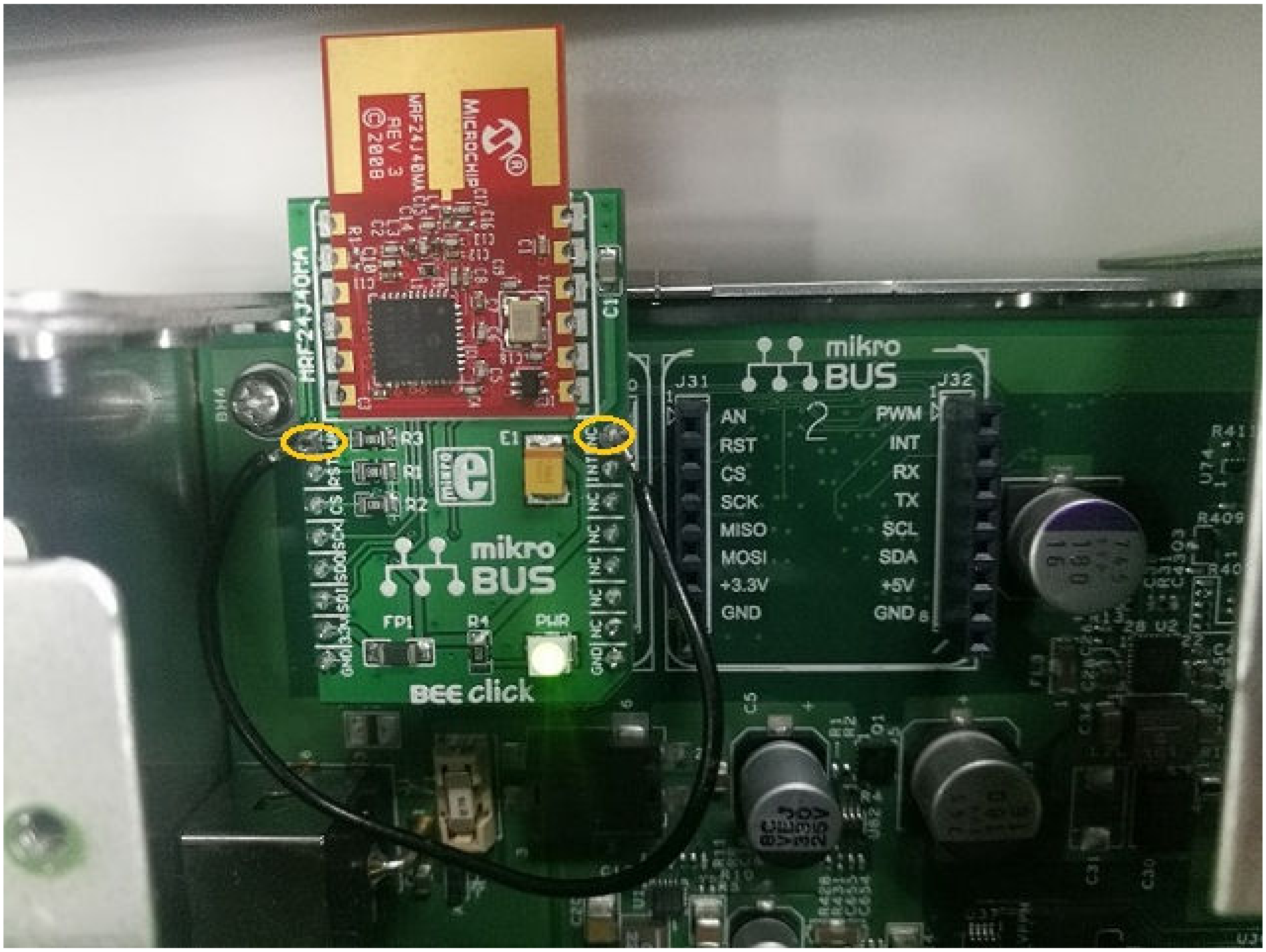


Figure 90. BEE Click Board hardware setup

NOTE

The WA pin of BEE Click Board connects with the NC pin.

15.4 Software preparation

In order to support BEE click board, use the following steps:

1. In OpenIL environment, use the command `make menuconfig` to enable the below options:

```
$make menuconfig
Target packages --->
  Hardware handling --->
    [*] i2c-tools
     Libraries --->
    Hardware handling --->
      [*] libbee
```

2. In Linux kernel environment, make sure the below options are enabled:

```
 $make linux-menuconfig
  Device Drivers --->
       SPI support --->
         <*>   Freescale DSPI controller
```

```
        <*> User mode SPI device driver support
    -*- GPIO Support  --->
    [*]   /sys/class/gpio/... (sysfs interface)
          Memory mapped GPIO drivers  --->
            [*] MPC512x/MPC8xxx/QorIQ GPIO support
```

NOTE

The above operation can be replaced by executing the command: `make nxp_ls1028ardbXXXX_defconfig`.

3. Use the `make` command to create the images.

15.5 Testing the BEE click board

The test application `bee_demo` is created by using the BEE Click Board library. This application can transfer the file between two BEE Click Boards.

1. You need to create a file in any path. For example, `./samples/test.txt`.
2. First, start a server node by running the command below:

```
bee_demo -s -f=XXX
```

The command parameters are as below:

- **-s**: This device node acts as a server.
- **-f=XXX**: This parameter is valid only on the server node. XXX is the file path (relative or absolute) to be transferred.

```
root@OpenIL-Ubuntu-LS1028ARDB:~# ls
samples
root@OpenIL-Ubuntu-LS1028ARDB:~# bee_demo -s -f=./samples/test.txt
spi mode: 0x0
bits per word: 8
max speed: 500000 Hz (500 KHz)
BEE  Click  Board  Demo.
This node is a server node.
Waiting for a client
Reading the content of the file
```

3. Start a client node on another LS1028ARDB by running the command `bee_demo -c`. In the above command, the parameter -c implies that this device node acts as a client. After receiving the file, the client node automatically exits. The received file is saved in the current path.

```
root@OpenIL-Ubuntu-LS1028ARDB:~# ls
                    samples
                    root@OpenIL-Ubuntu-LS1028ARDB:~# bee_demo -c
                    spi mode: 0x0
                    bits per word: 8
                    max speed: 500000 Hz (500 KHz)
                    BEE  Click  Board  Demo.
                    This node is a client node.
                    Starting to get a file
                    Send the SEQ_REQ command.
                    Send the SEQ_START command.
                    Send the SEQ_START command.
                    root@OpenIL-Ubuntu-LS1028ARDB:~# ls
                    samples  test.txt
                    root@OpenIL-Ubuntu-LS1028ARDB:~#
```

4. The following log is displayed to indicate that the server node finished sending a file.

```
Send the SEQ_INFO command.
Start to send the file
It's completed to send a file.
```

Chapter 16
BLE click board

This chapter introduces the features of the BLE P click board and how to use it on NXP's LS1028A reference design board (RDB)

16.1 Introduction

BLE P click carries the nRF8001 IC that allows you to add Bluetooth 4.0 to your device. The click communicates with the target board MCU through mikroBUS™ SPI (CS, SCK, MISO, MOSI), RDY and ACT lines, and runs on 3.3 V power supply.

BLE P click features a PCB trace antenna, designed for the 2400 MHz to 2483.5 MHz frequency band. The maximum device range is up to 40 meters in open space.

16.2 Features

Following are the features provided by BLE P clickboard:

- nRF8001 Bluetooth low energy RF transceiver
    - 16 MHz crystal oscillator
    - Ultra-low peak current consumption <14 mA
    - Low current for connection-oriented profiles, typically 2 µA
- PCB trace antenna (2400-2483.5 MHz, up to 40 meters)
- BLE Android app
- Interface: SPI (CS, SCK, MISO, MOSI), RDY and ACT lines
- 3.3 V power supply

16.3 Hardware preparation

Use the following hardware items for the BLE P click board demo setup:

1. LS1028ARDB
2. BLE P Click board
3. Android phone (option)

   The figure below depicts the hardware setup required for the demo:

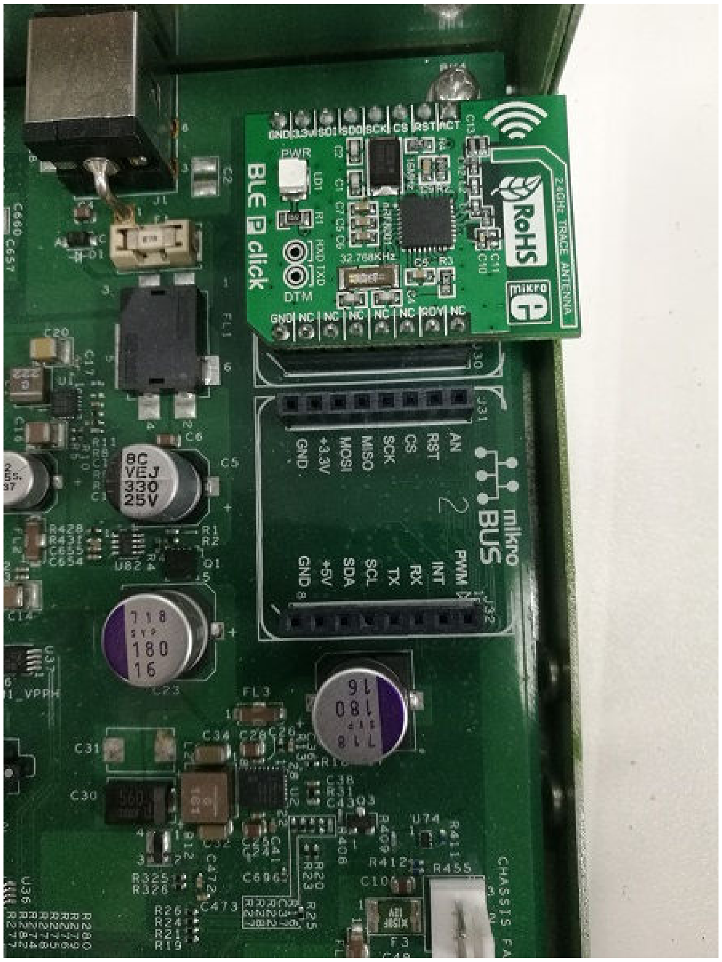


Figure 91. BLE P click board hardware setup

16.4 Software preparation

Use these steps for the BLE Pclickboard demo software setup:

- Download the JUMA UART (Android app) by using the link: https://apkpure.com/juma-uart/com.juma.UART
- Then, run the below steps in order to support BLE P click board:
    1. In OpenIL environment, use the command `make menuconfig` to enable the below options:

```
$make menuconfig
Target packages --->
       Hardware handling --->
           [*] i2c-tools
       Libraries --->
           Hardware handling --->
               [*] libblep
```

    2. In Linux kernel environment, make sure the below options are enabled:

```
$make linux-menuconfig
Device Drivers --->
      SPI support --->
```

```
        <*>   Freescale DSPI controller
        <*>   User mode SPI device driver support
```

3. Use the `make` command to create the images.

NOTE

The above operation can be replaced by executing the `make nxp_ls1028ardbXXXX_defconfig` file.

16.5 Testing the BLE P click board

Use the following steps for testing the BLE P click board:

1. Running the blep_demo application.

   The following log is displayed to indicate that the BLE P click board is initialized. At this time, you can scan for BLE P click board from your mobile phone or your computer's Bluetooth device. The name of the BLE P click board used is "MikroE"

```
root@OpenIL-Ubuntu:~# blep_demo
spi mode: 0x0
bits per word: 8
max speed: 500000 Hz (500 KHz)
Please input a command!
 Event device started: Setup
Error:no
Start setup command
.................................
Setup complete
 Event device started: Standby
 Advertising started : Tap Connect on the nRF UART app
Error:no
Send broadcast command successfully
```

2. Connection log

   Connect the BLE P click board via mobile app. On successful connection, the following log is displayed. Thereafter, the application can communicate with the BLE P click board.

```
Evt Connected
Evt Pipe Status
Evt link connection interval changed
ConnectionInterval:0x0006
SlaveLatency:0x0000
SupervisionTimeout:0x01F4
Evt Pipe Status
Evt link connection interval changed
ConnectionInterval:0x0027
SlaveLatency:0x0000
SupervisionTimeout:0x01F4
```

3. Disconnection log

Click the **Disconnect** button of the Android APP to disconnect from the BLE P click board. The following log displays that the disconnection is successful:

```
Evt Disconnected
Advertising started : Tap Connect on the nRF UART app Send broadcast command successfully
```

4. **Command line introduction**

The **blep _demo** application supports four command lines: **devaddr**, **name=**, **version**, and **echo**.

a. **devaddr**

This command is used to obtain the MAC address of the BLE P click board. You can run this command at any time.

```
devaddr
Please input a command!
Device address:DC:E2:6C:17:07:45
```

b. **name=**

This command is used to set the Bluetooth name of the ble p click board when broadcasting. No spaces are required after the equal sign "=", and the content after the equal sign is the set name. The maximum length is 16 characters.

```
name=ble_demo
Name set. New name: ble_demo, 8
Please input a command!
Unknow event:0x00
Set local data successfully
```

c. **version**

This command is used to obtain the version of the BLE P click board. You can run this command at any time.

```
version
Please input a command!
Unknow event:0x00
Device version
 Configuration ID:0x41
 ACI protocol version:2
 Current setup format:3
 Setup ID:0x00
 Configuration status:open(VM)
```

d. **echo**

This command is used to send a string to the Android app. This command should be executed after the connection is established. The maximum length is 20 characters.

The below log displays the message displayed after user tries to send a string when no connection is established:

```
echo hi
Please input a command!
Unknow event:0x00
 ACI Evt Pipe Error: Pipe #9
 Pipe Error Code: 0x83
 Pipe Error Data: 0x00
 Please connect the device before sending data
```

The below log is displayed when user sends a string after a connection is established:

```
echo hello,world!
Please input a command!
Unknow event:0x00
The number of data command buffer is 1
```

5. Receiving data

   When the Android app sends a string:

```
DataReceivedEvent: hi.yugxdr
```

Chapter 17 QT

This chapter introduces the QT feature for OpenIL and provides instructions on how to enable this feature on NXP's LS1028A reference design board.

17.1 Introduction

Qt is a full development framework with tools designed to streamline the creation of applications and user interfaces for desktop, embedded, and mobile platforms. For details, see http://doc.qt.io/qt-5/index.html

This section describes how to enable QT5 in OpenIL.

17.2 Software settings and configuration

Use the following steps to configure QT5 on target board and build the images.

1. **Configure the target board**: There are two configuration files used for LS1028ARDB board: `nxp_ls1028ardb-64b_defconfig` and `nxp_ls1028ardb-64b_ubuntu_defconfig`. Use one of them:

   ```
   make nxp_ls1028ardb-64b_defconfig
   ```

   or

   ```
   make nxp_ls1028ardb-64b_ubuntu_defconfig
   ```

2. **Configure QT5**: Use the command `make menuconfig` to configure the QT5:

   ```
   Target packages ->
   Graphic libraries and applications (graphic/text) ->
   [*] Qt5 ->
         [*] Compile and install examples (with code)
         [*] concurrent module
         [*] MySQL Plugin
         [*] PostgreSQL Plugin
         [*] gui module
         [*] widgets module
         [*] fontconfig support
         [*] GIF support
         [*] JPEG support
         [*] PNG support
         [*] DBus module
         [*] Enable ICU support
         [*] Enable Tslib support
         [*] qt5enginio
         [*] qt5imageformats
         [*] qt5location
         [*] qt5multimedia
         [*] qt5quickcontrols
         [*] qt5quickcontrols2
         [*] qt5sensors
         [*] qt5serialbus
         [*] qt5svg
   ```

3. Build the images using the command:

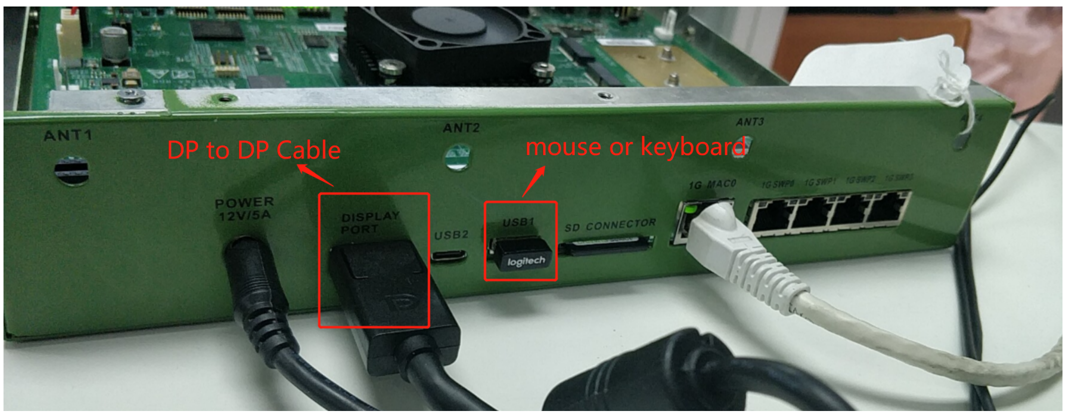
```
make -j8
```

17.3 Hardware setup

For the QT setup, you require the following hardware:

1. Monitor that supports DP interface
2. Cable matters DisplayPort to DisplayPort (DP to DP Cable)
3. USB wired/wireless mouse or keyboard

Figure 92. Hardware setup for QT



17.4 Running the QT5 demo

This section describes the steps for configuring the environment and running the Qt demos for LS1028ARDB.

17.4.1 Environment setting

Use the steps listed below to configure the environment settings:

1. For `nxp_ls1028ardb-64b_defconfig` configuration
   - Make sure that the `fonts` directory exists in the `/usr/share/` directory. If it does not exist, you can find it in the `root` directory, and copy one or more to `/usr/share`, as shown in the example below:

```
[root@OpenIL:~]# cd /
[root@OpenIL:]# find ./ -name fonts
./usr/lib/qt/examples/quickcontrols2/texteditor/fonts
./usr/lib/qt/examples/quickcontrols2/swipetoremove/fonts
./usr/lib/qt/examples/quick/text/fonts
./usr/lib/qt/examples/quick/text/fonts/content/fonts
./usr/lib/qt/examples/quickcontrols/extras/dashboard/fonts
./usr/lib/qt/examples/quickcontrols/extras/gallery/fonts
./usr/share/imlib2/data/fonts
./usr/share/fonts
./usr/share/fonts/content/fonts
./etc/fonts
[root@OpenIL:]# cp -r /usr/lib/qt/examples/quick/text/fonts /usr/share/
[root@OpenIL:]#
```

2. For `nxp_ls1028ardb-64b_ubuntu_defconfig` configuration

- Replace `/lib/aarch64-linux-gnu/libz.so.1` with the following commands:

```
rm /lib/aarch64-linux-gnu/libz.so.1
ln -s -f /usr/lib/libz.so.1.2.11 /lib/aarch64-linux-gnu/libz.so.1
```

- Make sure that the `fonts` directory exists in the `/usr/share/` directory. If it does not exist, you can find it in the `root` directory and copy one or more to `/usr/share/` using the following commands:

```
cd /
find ./ -name fonts
cp -r /usr/lib/qt/examples/quick/text/fonts/content/fonts /usr/share
```

```
root@OpenIL-Ubuntu-LS1028ARDB:/# find ./ -name fonts
./usr/lib/qt/examples/quick/text/fonts
./usr/lib/qt/examples/quick/text/fonts/content/fonts
./usr/lib/qt/examples/quickcontrols2/swipetoremove/fonts
./usr/lib/qt/examples/quickcontrols2/texteditor/fonts
./usr/lib/qt/examples/quickcontrols/extras/gallery/fonts
./usr/lib/qt/examples/quickcontrols/extras/dashboard/fonts
./etc/fonts
root@OpenIL-Ubuntu-LS1028ARDB:/# cp -r /usr/lib/qt/examples/quick/text/fonts/content/fonts /usr/share/
```

The QT5 framework is configured now, and user can add any applications.

17.4.2 Running the demos

There are many sample demos in the directory `/usr/lib/qt/examples`. Following are some of the demos and their corresponding commands:

1. **Example1:** `/usr/lib/qt/examples/widgets/widgets/wiggly/wiggly --platform linuxfb`

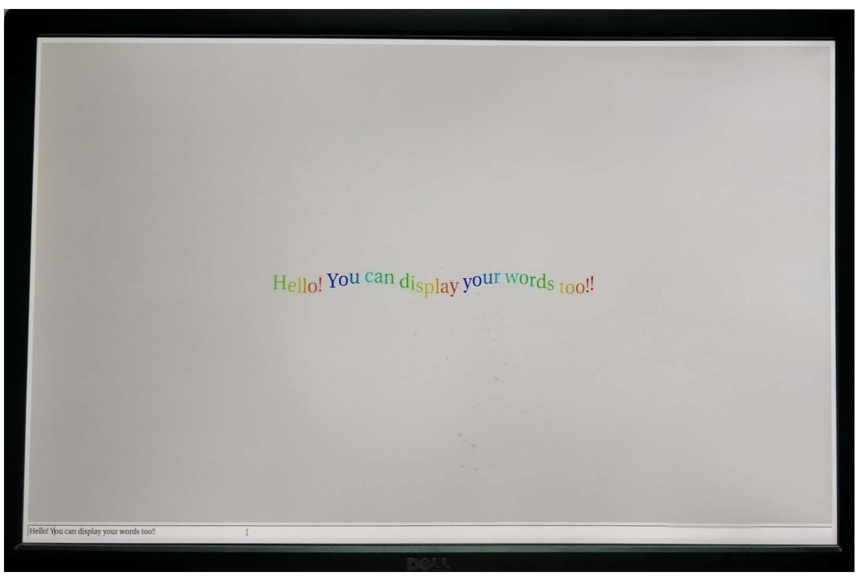


Figure 93. Example 1: Wiggly text

2. **Example 2:** `/usr/lib/qt/examples/quick/demos/clocks/clocks --platform linuxfb`

Figure 94. Example 2: Clocks

3. **Example 3:** `/usr/lib/qt/examples/gui/analogclock/analogclock --platform linuxfb`

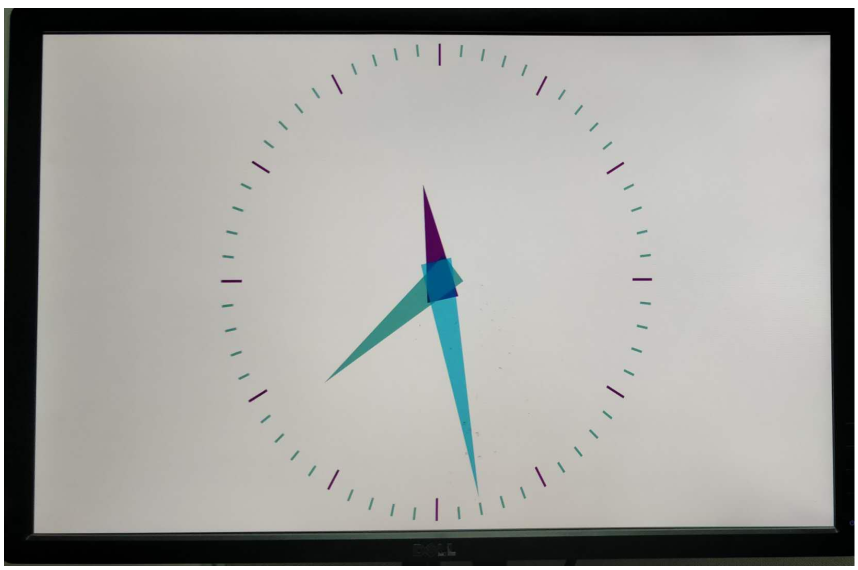


Figure 95. Example 3: Analog clock

Chapter 18
EdgeScale client

This chapter describes edgescale, its features and the procedure to use Edgescale on NXP supported hardware platforms.

18.1 What is EdgeScale

EdgeScale is a unified, scalable, and secure device management solution for Edge Computing applications. It enables OEMs and developers to leverage cloud compute frameworks like AWS Greengrass, Azure IoT and Aliyun on Layerscape devices. It provides the missing piece of device security and management needed for customers to securely deploy and manage a large number of Edge computing devices from the cloud. End-users and developers can use the EdgeScale cloud dashboard to securely enroll Edge devices, monitor their health, attest and deploy container applications and firmware updates.

EdgeScale can also be used as a development environment to build containers and generate firmware.

18.2 Edgescale features

Following are the features supported by Edgescale:

- EdgeScale dashboard for users
- Secure device enrolment
- Secure key/certificate provisioning
- OTA: firmware update (LS1012A, LS1043. LS1046, or LS1028)
- Device status monitoring on the cloud
- Dynamic deployment of container-based applications
- The above specified features are currently supported in LSDK. For more details, please visit: EDGESCALE: EdgeScale for Secure Edge Computing

18.3 Building EdgeScale client

To Build the EdgeScale client in OpenIL for LS1043A, LS1046A, and LS1028A, follow the configuration below:

```
Make menuconfig
Target packages  --->
Edge-scale service  --->
[*] qorio edgescale eds
[*] qorio eds kubelet
[*] qorio eds bootstrap
```

18.4 Procedure to start EdgeScale

For complete details on how to start EdgeScale, visit the URL https://doc.edgescale.org/.

NOTE

Follow these steps after downloading the device identification info file (which is a script file):

1. Copy the script file to the DUT and run it using the command below:

```
sh xxxx.sh /dev/mmcblk0
```

2. Then, reboot the board.
3. Run the below command to start edgescale client in Linux prompt:

```
sh /usre/local/edgescale/bin/startup.sh
```

Chapter 19
Known issues

The following table lists the known issues for Open Industrial Linux for this release.

Table 45. 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 20
Revision history

The table below summarizes revisions to this document.

Table 46. Document revision history

Date	Document version	Topic cross- reference	Change description
20/02/20	1.7.1	Operation examples	Updated this section.
17/01/20	1.7	nxp-servo	Added the chapter.
		IEEE 1588/802.1AS	Added the chapter
		LX2160ARDB	Added the section.
		Getting Open IL	Updated the section.
		NETCONF/YANG	Other updates.
31/08/19	1.6	Using TSN features on LS1028ARDB	• Information related to pcpmap command removed from the section Basic TSN configuration examples on ENETC and Basic TSN configuration examples on the switch. • Port names "eno/swp0" changed to "swp0" for few tsntool commands. • Note added in section Stream identification for usage of `nulltagged` and `streamhandle` parameters. • Added the section TSN stream identification. • Other minor updates.
		Table 4	Updated the table "Host system mandatory packages". Added `autogen autoconf libtool` and `pkg-config` packages.
		BEE Click Board	Added this chapter.
		Web UI demo	Added this section in NETCONF/YANG.
		NETCONF/YANG	• Added the section Enabling NETCONF feature in OpenIL and other updates.
01/05/2019	1.5	Interface naming	Added the section. Describes interface naming for U-Boot and Linux for LS1028ARDB.
		Using TSN features on LS1028ARDB	Updated this section in the Chapter TSN .
		BLE click board	Added the Chapter.
		EdgeScale client	Added the Chapter.
		Getting Open IL	Updated the OpenIL version and Git tag.
01/02/2019	1.4	Supported NXP platforms	Added support for LS1028ARDB (64-bit and Ubuntu). Updated various sections accordingly.

Table continues on the next page...

Table 46. Document revision history (continued)

Date	Document version	Topic cross- reference	Change description
		Getting Open IL	Updated the OpenIL version and Git tag.
		LS1028ARDB	Added this Section for LS1028ARDB support.
		TSN	Reorganized this Chapter and added separate Section for Using TSN features on LS1028ARDB.
		NFC click board	Added the Chapter.
		FlexCAN	Minor updates in this Chapter. Also added the section, Hardware preparation for LS1028ARDB and Testing CAN bus.
		QT	Added the Chapter.
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 and updated Basic setup. Updates in other sections.
31/05/2018	1.2	Hardware requirements	Updated the Section, "Hardware requirements" for RTnet.
		Software requirements	Updated the Section, "Software requirements" for RTnet.
18/04/2018	1.1.1	RTnet	Added the Section, "RTnet".
		Switch settings	Added a note for LS1043A switch setting.
30/03/2018	1.1	Supported industrial features	Added support for industrial IoT baremetal framework in this section.
		Booting up the board	Added a note for steps to be performed before booting up the board.
		Reference documentation	Added the section.
22/12/2017	1.0	OPC UA	Added the Chapter.
		TSN	Chapters for "1-board TSN demo" and "3-board TSN demo" replaced by a single chapter, "TSN demo".
		IEEE 1588	• 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	OTA - Xenomai Cobalt 64-bit and SJA1105 support added.
		TSN	Qbv support added.
		SELinux	SELinux support for LS1043 / LS1046 Ubuntu Userland added.

Table continues on the next page...

Table 46. Document revision history (continued)

Date	Document version	Topic cross- reference	Change description
		OP-TEE	OP-TEE support for LS1021ATSN platform added.
		4G-LTE Modem	4G LTE module - 64-bit support for LS1043ARDB, LS1046ARDB, and LS1012ARDB added.
		NXP OpenIL platforms	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

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 02/2020

Document identifier: OpenILUG

arm