User Manual

for Industry Linux Solution

Release-v0.1

Table of Contents

1 INTRODUCTION 1
1.1 PURPOSE 1
1.2 SCOPE 1
1.3 REFERENCES 1
2 GETTING STARTED 2
2.1 INSTALL YOCTO 2
2.2 SET UP HOST ENVIRONMENT 2
2.3 SET UP POKY 3
2.4 PERFORM BUILDS 4
3 LS1021ATSN PLATFORM 6
3.1 INTRODUCTION 6
3.2 SWITCH SETTINGS 6
3.3 BUILDING IMAGES 7
3.4 ENVIRONMENT PREPARE 8
3.4.1 Basic Host Set-up 8
3.4.2 Target Board Startup 9
3.5 SD CARD PREPARE 10
3.6 U-BOOT ENVIRONMENT VARIABLES 11
3.7 SYSTEM MEMORY MAP 11
3.8 PROGRAMMING A NEW U-BOOT 12
3.8.1 RCW (Reset Configuration World) 12
3.8.2 Programming U-boot to SD card 12
3.9 DEPLOYMENT 12
3.9.1 SD Deployment 12
3.9.2 Ramdisk Deployment from TFTP 13
3.9.3 NFS Deployment 14
4 LS1043ARDB AND LS1046ARDB PLATFORM 16
4.1 LS1043ARDB PLATFORM 16
4.1.1 Introduction 16
4.1.2 Building Images 16
4.1.3 Reference 16

4.2 LS1046ARDB PLATFORM 16
4.2.1 Introduction 16
4.2.2 Building Images 17
4.2.3 Reference 17
4.3 ARMV8 AARCH32 MODE 17
5 XENOMAI RT LINUX 19
5.1 XENOMAI INTRODUCTION 19
5.2 XENOMAI RUNNING MODE 19
5.3 BUILDING KERNEL IMAGE FOR XENOMAI MERCURY MODE 19
5.4 BUILDING XENOMAI MERCURY TOOLS 20
5.5 RUNNING XENOMAI MERCURY 20
6 TSN DEMO 21
6.1 INTRODUCTION 21
6.2 DEMO OVERVIEW 21
6.3 USE OF VLAN TAGS IN THE DEMO 22
6.4 SETUP PREPARATION 24
6.5 SJA1105 SWITCH CONFIGURATIONS 26
6.6 ANALYSIS 27
6.6.1 Standard configuration 27
6.6.2 Policing configuration 28
6.6.3 Scheduling configuration 30
7 IEEE 1588 32
7.1 INTRODUCTION 32
7.2 PTP DEVICE TYPES 32
7.3 LINUXPTP STACK 33
7.3.1 Introduction 33
7.3.2 Features 33
7.4 PTPD STACK 33
7.4.1 Introduction 33
7.4.2 Features 33
7.5 QUICK START GUIDE 33
7.5.1 Requirement 33
7.5.2 Ethernet interfaces connection 34

7.5.3 PTP stack startup ... 35
7.6 KNOWN ISSUES AND LIMITATIONS ... 36
7.7 LONG TERM TEST RESULTS ... 36
8 KNOWN ISSUES ... 39
APPENDIX A. VERSION TRACKING ... 40

1 Introduction

1.1 Purpose

This document serves as a User Manual for the Industry Linux Solution v0.1 release.

1.2 Scope

The scope of this document is limited to the Industry Linux Solution v0.1 release

1.3 References

Document	Version	Location
QorIQ SDK online Document	V2.0-1611	https://freescale.sdlproducts.com/LiveContent/web/pub.xql?c=t&action=home&pub=QorIQ_SDK&lang=en-US

2 Getting Started

2.1 Install Yocto

How to install Yocto Project on the host machine.

1. Mount the ISO on your machine:

$ sudo mount -o loop Industry-Linux-Solution-Release-<version>-SOURCE-<yyyymmdd>-yocto.iso /mnt/cdrom

2. As a non-root user, install Yocto Project:
$ /mnt/cdrom/install

3. When you are prompted to input the install path, ensure that the current user has the correct permission for the install path.

There is no uninstall script. To uninstall Yocto Project, you can remove the <yocto_install_path>/Industry-Linux-Solution-Release-xxxx directory manually.

Note:

• The source ISO contains the package source tarballs and Yocto Project recipes. It can be installed and used to do non-cache build.

• The cache ISO contains the pre-built cache binaries. To avoid a long time build, you can install the source ISO and the cache ISO in the same installation folder.

• The image ISO includes all prebuilt images: flash images, standalone toolchain installer, HD rootfs images and small images.

• The source ISO can be used separately. The core ISO and the source ISO should work together.

2.2 Set Up Host Environment

The following is the detailed package list on the CentOS hosts:

```
$ sudo yum install gawk make wget tar bzip2 gzip python unzip perl patch \
    diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath socat SDL-devel xterm
```

For the Fedora hosts:

```
$ sudo yum install gawk make wget tar bzip2 gzip python unzip perl patch \
    diffutils diffstat git cpp gcc gcc-c++ glibc-devel texinfo chrpath \
    ccache perl-Data-Dumper perl-Text-ParseWords perl-Thread-Queue socat \
    findutils which SDL-devel xterm
```

For Ubuntu and Debian hosts:

*$ sudo apt-get install gawk wget git-core diffstat unzip texinfo gcc-multilib *

build-essential chrpath socat libsdl1.2-dev xterm

Extra packages are needed for Ubuntu-64b:

$ sudo apt-get install lib32z1 lib32ncurses5 lib32bz2-1.0 ia32-libs lib32ncurses5-dev

For OpenSUSE host:

$ sudo zypper install python gcc gcc-c++ libtool subversion git chrpath automake make wget diffstat makeinfo freeglut-devel libSDL-devel

2.3 Set Up Poky

Source the following poky script to set up your environment for your particular NXP platform. This script needs to be run once for each terminal, before you begin building source code.

$ . ./fsl-setup-env -m <machine>

For example:

$ . ./fsl-setup-env -m ls1043ardb-32b

The following shows the usage text for the fsl-setup-env command:

$ . ./fsl-setup-env -h

Usage: . fsl-setup-env -m <machine>

Supported machines:

ls1021atsn ls1043ardb-32b ls1046ardb-32b

Optional parameters:

** [-m machine]: the target machine to be built.*

** [-b path]: non-default path of project build folder.*

** [-j jobs]: number of jobs for make to spawn during the compilation stage.*

** [-t tasks]: number of BitBake tasks that can be issued in parallel.*

** [-d path]: non-default path of DL_DIR (downloaded source)*

** [-c path]: non-default path of SSTATE_DIR (shared state Cache)*

** [-g]: enable Carrier Grade Linux*
** [-l]: lite mode. To help conserve disk space, deletes the building directory once the package is built.*
** [-h]: help*

2.4 Perform Builds

How to Set Up a Cross Compile Environment and Perform Builds

Follow these steps to do builds using Yocto Project. Be sure to set up the host environment before doing these steps.

1. Change to build machine directory

$ cd <sdk-install-dir>/build_<machine>

2. Build target

$ bitbake <image-target>

Where <image-target> is one of the following:

- fsl-image-minimal: contains basic packages to boot up a board
- fsl-image-core: contains common open source packages and NXP specific packages.
- fsl-image-full: contains all packages in the full package list.
- fsl-image-kernelitb: A FIT image comprising the Linux image, dtb and rootfs image.
- fsl-image-mfgtool: contains all the user space apps needed to deploy the fsl-image-mfgtool image to a USB stick, hard drive, or other large physical media.
- fsl-image-virt: contains toolkit to interact with the virtualization capabilities of Linux
- core-image-x11: NXP image with a very basic X11 image with a terminal
- fsl-toolchain: the cross compiler binary package
- package-name(xenomai): build a specific package

Contents of the Built Images Directory:

A Yocto Project build produces images that will be located in the following directory:

<sdk-install-dir>/build_<machine>/tmp/deploy/images/<machine>/

The following list shows the typical directory/image files (exact contents depend on the setting of the <IMAGE_FSTYPES> variable):

- *fsl-image-<machine>.ext2.gz.u-boot - ramdisk image that can be loaded with U-Boot*
- *fsl-image-<machine>.ext2.gz - gzipped ramdisk image*
- *fsl-image-<machine>.tar.gz - gzipped tar archive of the image*
- *uImage-<machine>.bin - kernel binary of the image*
- *u-boot-<machine>.bin - U-Boot binary image that can be programmed into board Flash*
- *uImage-<machine>.dtb - device tree binary (dtb).*
- *kernel-fsl-<machine>.itb – ITB image.*
- *hv/hv.uImage - uImage for hypervisor*

- *rcw/*/rcw_*.bin - rcw*

Note: For additional Yocto Project usage information, please refer to the https://www.yoctoproject.org/.

3 LS1021ATSN Platform

3.1 Introduction

LS1021ATSN-PA board is a low cost, full featured, open source next generation IOT gateway reference design supporting a broad array of IoT applications including building/home management, smart cities and networked industrial services. The feature rich IOT gateway reference design based on QorIQ LS1021A embedded processor.

This chapter provides board-specific configuration and instructions for different methods of deploying U-Boot, Linux kernel and root file system to the target board. The guide starts with generic host and target board pre-requisites. This is followed by board-specific configuration:

• Switch Settings

• U-Boot environment Variables

• System Memory Map

The board supported two boot modes by configured switch settings.

• SD Boot

• QSPI Boot (unsupported in v0.1)

Once the board is set-up and configured appropriately, building images and select one of the following deployment methods:

• Ramdisk deployment from TFTP

• SD deployment

• NFS deployment

3.2 Switch Settings

The following table lists and describes the switch configuration for LS1021ATSN-PA Board.

Switch	**Settings**(OFF=1,ON=0)	**Options**	**Description**
SW2.1	OFF	EXP1_GPIO8	0-Logic LOW in GPIO, 1 Logic HIGH
SW2.2	OFF	EXP1_GPIO9	0-Logic LOW in GPIO, 1 Logic HIGH
SW2.3	OFF	EXP1_GPIO10	0-SJA1105 Switch Soft Reset, 1 Logic HIGH

SW2.4	OFF	MBED_SWD_SW_EN	Reserved
SW2.5	OFF	BOOTSEL_QSPI_SD	0 - QSPI, 1 – MicroSDHC
SW2.6	OFF	SYSTEM_CLK_SEL	0 - Diff-Sysclk, 1 – Sysclk

The following table lists the jumper settings.

Jumpers	**Default settings on LS1021ATSN-PA**	**Description**
H5	OFF	VDD_LP Source Select OFF- Battery ON- +12V_HS

3.3 Building Images

Follow these steps to do builds using Yocto Project. Be sure to set up the host environment before doing these steps.

1. *$. ./fsl-setup-env -m ls1021atsn*
2. *$ cd <yocto_install_path>/build_ls1021atsn/*
3. *$ bitbake <image-target>*

Where <image-target> is one of the following:

• fsl-image-minimal: contains basic packages to boot up a board

• fsl-image-core: contains common open source packages and FSL specific packages.

• fsl-image-full: contains all packages in the full package list.

• fsl-image-virt: contains toolkit to interact with the virtualization capabilities of Linux

• fsl-toolchain: the cross compiler binary package

Contents of the Built Images Directory:

A Yocto build produces images that will be located in the following directory:

<yocto_install_patch>/build_ls1021atsn/tmp/deploy/images/ls1021atsn/

The following list shows the typical directory/image files (exact contents depend on the setting of the <IMAGE_FSTYPES> variable):

• fsl-image ls1021atsn.ext2.gz.u-boot - ramdisk image that can be loaded with U-Boot

• fsl-image- ls1021atsn.ext2.gz - gzipped ramdisk image

- fsl-image- ls1021atsn.tar.gz - gzipped tar archive of the image
- uImage-ls1021atsn.bin - kernel binary of the image
- u-boot ls1021atsn.bin - U-Boot binary image that can be programmed into board Flash
- uImage- ls1021atsn.dtb - device tree binary (dtb).

3.4 Environment Prepare

3.4.1 Basic Host Set-up

Since TFTP will be used to download files onto the target board, a TFTP server must be running on your host system. If you are going to use NFS deployment then an NFS server must also be running on your host system.

Once TFTP and NFS servers are installed, use the following generic instructions to complete the host set-up:

1. Create the tftpboot directory.

   $ mkdir /tftpboot

2. Copy over kernel, bootloader, and flash filesystem images for your deployment to the /tftpboot directory:

   $ cp <yocto_work_dir>/build_ls1021atsn/tmp/deploy/images/ /tftpboot*

3. Use Yocto to generate a tar.gz type file system, and uncompress it in <nfs_root_path>.

4. Edit /etc/exports and add the following line:

   <nfs_root_path> <target_board_IP> (rw,no_root_squash, async)

5. Edit /etc/xinetd.d/tftp to enable TFTP server:

```
service tftp
{
  disable= no
  socket_type= dgram
  protocol= udp
  wait= yes
  user= root
  server= /usr/sbin/in.tftpd
  server_args= /tftpboot
}
```

6. Restart the nfs and tftp servers on your host:

   $ /etc/init.d/xinetd restart
   $ /etc/init.d/nfs restart

7. Connect board to the network.

8. Connect the target to the host via a cross cable serial connection.

9. Open a serial console tool on the host system and set it up to talk to the target board:

   - Select appropriate serial device.

- Configure the serial port with the following settings: Baud rate = 115,200; Data= 8 bit; Parity = none; Stop = 1 bit; Flow control = none.
- Power on board and see the console prompt.

NOTE

1. The Linux distribution running on your host will determine the specific instructions to use.
2. Steps 3 and 4 are only necessary when using NFS deployment.

3.4.2 Target Board Startup

The LS1021ATSN-PA comes with a Micro SDHC card preloaded with the U-Boot and Linux images. Serial connectivity for the LS1021ATSN-PA is provided through the Micro USB connector on the front side.

Before starting up the board, please make sure mbed serial access driver has been installed on your Windows PC (the mbed serial port works by default on Linux) correctly. If not, please follow up below link:

https://developer.mbed.org/handbook/Windows-serial-configuration

To start up the board:

1. Connect the PC with LS1021ATSN-PA micro-USB console port using a Micro-B USB cable.
2. Insert the Micro SDHC card into the Micro SDHC card slot located on the front side of the LS1021ATSN-PA board. Note that the Micro SDHC card contacts should be facing down.
3. Set up a serial terminal using a PC communication program such as TeraTerm set to 115200-8-N-1.
4. Plug in the power supply barrel into the port labeled '12V power' located on the rear side.
5. Plug the power supply into the mains and switch on.
6. Check if you can see U-Boot in the terminal window.
7. Below is an example of a typical U-Boot log:

```
U-Boot 2016.09-16646-g2c8ab99-dirty (Dec 08 2016 - 15:38:46 +0800)

CPU:   Freescale LayerScape LS1021E, Version: 2.0, (0x87081120)
Clock Configuration:
       CPU0(ARMV7):1200 MHz,
       Bus:300  MHz, DDR:800  MHz (1600 MT/s data rate),
Reset Configuration Word (RCW):
       00000000: 0608000c 00000000 00000000 00000000
       00000010: 30000000 08007900 60040a00 21046000
       00000020: 00000000 00000000 00000000 20002000
       00000030: 20024800 8804b340 00000000 00000000
Model: LS1021A TSN Board
Board: LS1021ATSN
I2C:   ready
DRAM:  1 GiB
```

Using SERDES1 Protocol: 48 (0x30)
MMC: FSL_SDHC: 0
EEPROM: Invalid ID (ff ff ff ff)
In: serial
Out: serial
Err: serial
SEC0: RNG instantiated
SATA link 0 timeout.
AHCI 0001.0300 1 slots 1 ports ? Gbps 0x1 impl SATA mode
flags: 64bit ncq pm clo only pmp fbss pio slum part ccc
Found 0 device(s).
SCSI: Net: eTSEC1 is in sgmii mode.
eTSEC2 is in sgmii mode.
PCIe0: pcie@3400000 Root Complex: no link
PCIe1: pcie@3500000 disabled
eTSEC1 [PRIME], eTSEC2

3.5 SD Card Preparation

If there is a new Micro SDHC card without any images in it. You need to prepare the Micro SDHC card:

1. Confirm the capacity of SD card is at least 2GB.

2. Insert the Micro SDHC card into a SDHC card reader and plug in host PC. Check the device node of SD card (for example, identified as /dev/sdb).

3. Load ls1021atsn_sd.img into the SD card on host PC.

   mount Industry-Linux-Solution-Release-V0.1-CORTEXA7-IMAGE-xxx-yocto.iso /media/isofile

   cd /media/isofile/sdimage

   dd if=ls1021atsn_sd.img of=/dev/sdb

   Note: Please use root node of SDcard (/dev/sdb) instead of partition node, like /dev/sdb1, /dev/sdb2 or /dev/sdbN.

4. After writing the image into SD card, do as above mentioned to start up the board and write u-boot ENV.

5. The mapping address of the Micro SDHC card is showing in the following table:

Start Physical Address	End Physical Address	Memory Type	Size
0x0000_0000	0x0000_0FFF	Partition Table	4K
0x0000_1000	0x000F_FFFF	U-boot	1020K
0x0010_0000	0x001F_FFFF	U-boot ENV	1M
0x0020_0000	0x191F_FFFF	Partition 1(File System)	400M
0x1920_0000	0xFFFF_FFFF	Remainder	-

If you want to update u-boot, kernel or filesystem, see section 7. "Programming a new u-boot", and section 8. "Deployment"

NOTE

U-Boot is configured to automatically load the Linux kernel, device tree, and file system binaries from the SD card to memory and boot to a Linux prompt. When prompted for a login, type root and press enter. The board power cannot be unplugged when linux system is running, use poweroff command to shutdown linux system.

3.6 U-Boot Environment Variables

To support TFTP based deployments, set up the U-Boot environment once, and save it, so that settings persist on subsequent resets.

```
=>setenv ipaddr <board_ipaddress>
=>setenv serverip <tftp_serverip>
=>setenv gatewayip <your_gatewayip>
=>setenv ethaddr <mac addr0>
=>setenv eth1addr <mac addr1>
=>setenv eth2addr <mac addr2>
=>setenv ethprime <ethx>
=>setenv ethact <ethx>
=>setenv netmask 255.255.x.x
=>saveenv
```

3.7 System Memory Map

In 32-bit u-boot, there is a 1:1 mapping of physical address and effective address. After system startup, the boot loader maps physical address and effective address as shown in the following table:

Start Physical Address	End Physical Address	Memory Type	Size
0x0100_0000	0x0FFF_FFFF	CCSR	240MB
0x1000_0000	0x1000_FFFF	OCRAM0	64KB
0x1001_0000	0x1001_FFFF	OCRAM1	64 KB
0x2000_0000	0x20FF_FFFF	DCSR	16MB
0x4000_0000	0x5FFF_FFFF	QSPI	512MB
0x8000_0000	0xFFFF_FFFF	DDR	2GB

3.8 Programming a New U-Boot

3.8.1 RCW (Reset Configuration World)

For SD boot, RCW configure file is integrated in u-boot source. When building u-boot, RCW can be built and integrated into u-boot-sdcard.bin. If you want to update RCW, update cfg file and rebuild u-boot, then programming u-boot to sd card.

The file path is board/freescale/ls1021atsn/ls102xa_rcw_sd_qspi.cfg.

3.8.2 Programming U-boot to SD card

To program U-boot, first boot the board to u-boot. Next, load the new u-boot SD boot image (u-boot-sdcard.bin) to RAM by downloading it via TFTP and then copying it to SD card with blk offset 0x8. Execute the following commands at the U-Boot prompt to program the RCW to flash and reset to alternate bank.

```
=>tftp 81000000 u-boot-sdcard.bin
=>mmc erase 8 0x500
=>mmc write 0x81000000 8 0x500
```

On host Linux, you can use following command to write u-boot into SD card (for example, identified as /dev/sdb with USB card reader).

```
$ dd if=u-boot-sdcard.bin of=/dev/sdb bs=512 seek=8
```

Note: u-boot-sdcard.bin is the u-boot binary located at:

<yocto_install_patch>/build_ls1021atsn/tmp/deploy/images/ls1021atsn/

3.9 Deployment

3.9.1 SD Deployment

The ext2 file system generated by Yocto could be stored in a SD card.

3.9.1.1 Deploy Filesystem to the SD Card

Once the U-Boot network parameters have been set, follow the steps below to deploy the filesystem to the SD card. Please note that the SDcard size should be at least 2GB:

1. Connect the card reader with SD card to the Linux Host PC (for example, identified as sdb).

2. Create two partitions by "fdisk /dev/sdb", first partition is for MS-DOS (size 2MB), and the other is for ext2 (the rest capacity of this SD card).

```
$ fdisk /dev/sdb
```

3. Use the mkfs.ext2 command to create the filesystem.

```
$ mkfs.vfat /dev/sdb1
$ mkfs.ext2 /dev/sdb2
```

4. Create temporary directory in host PC and mount the ext2 partition to the temp.

```
$ mkdir /temp
$ mount /dev/sdb2 /temp
$ cd /temp
```

5. Copy the file system to harddisk by extracing the fsl-image-full-<board>-<release date>-rootfs.tar.gz. Remove the tarball after extracting rootfs.

```
$ cp fsl-image-full-<board>-<release date>-rootfs.tar.gz  ./
$ sudo tar -zxvf fsl-image-full-<board>-<release date>-rootfs.tar.gz
$ rm fsl-image-full-<board>-<release date>-rootfs.tar.gz
```

6. Make sure the kernel image and dtb file are in /temp/boot directory, then umount the /temp.

```
$ cp uImage and uImage.dtb to /temp/boot
$ umount /temp
```

7. Plug in the SD card to the target board and power on.

3.9.1.2 Setting U-Boot Environment

You can place the ext2 filesystem and kernel on the SD card, then the kernel can boot up automatically after the board is powered on or after reset. Prior to this deployment, make sure U-Boot parameters have been set up as follows:

```
=> setenv bootfile uImage
=> setenv fdtfile uImage.dtb
=> setenv bootcmd 'setenv bootargs root=/dev/mmcblk0p2 rw rootdelay=5 console=$consoledev,
$baudrate;mmcinfo;ext2load mmc 0:2 $loadaddr /boot/$bootfile;ext2load mmc 0:2 $fdtaddr /
boot/$fdtfile;bootm $loadaddr - $fdtaddr'
=> save
```

3.9.2 Ramdisk Deployment from TFTP

3.9.2.1 Setting U-Boot Environment

The images generated by Yocto allow you to perform ramdisk deployment. Before performing ramdisk deployment, the U-Boot environment variables need to be configured.

Refer to U-Boot Environment Variables chapter to set the U-Boot environment variables. In addition, execute the following commands at the U-Boot prompt to prepare for ramdisk deployment from TFTP:

```
=>setenv bootargs ‘root=/dev/ram0 rw ramdisk_size=40000000 console=ttyS0,115200’
=>saveenv
```

NOTE

ramdisk_size needs to be set if the ramdisk uncompress file size is bigger than default setting. It should be more than ramdisk uncompress file size. The file size information is printed in Yocto build log.

3.9.2.2 Booting Up the System

Execute the following commands to TFTP the images to the board, then boot into Linux.

=>tftp 83000000 < uImage_name>
=>tftp 88000000 fsl-image-core-<platform>.ext2.gz.u-boot
=>tftp 8f000000 <platform_dtb_name>
=>bootm 83000000 88000000 8f000000

Now the board will boot into Linux using the images generated by Yocto.

3.9.3 NFS Deployment

3.9.3.1 Generating File System with Yocto

Use Yocto to generate an fsl-image-full-<board>-<release date>-rootfs.tar.gz file system, and uncompress it for NFS deployment.

3.9.3.2 Setting Host NFS Server Environment

a. On the Linux host NFS server, add the following line in the file /etc/exports:

nfs_root_path board_ipaddress(rw,no_root_squash,async)

b. Restart the NFS service:

/etc/init.d/nfs restart

NOTE

nfs_root_path: the NFS root directory path where file system just is uncompressed to on NFS server.

3.9.3.3 Setting U-Boot Environment

The NFS file system generated by Yocto allows you to perform NFS deployment. Before performing NFS deployment, the U-Boot environment variables need to be configured. Refer to Configuring U-Boot Network Parameters to set the U-Boot environment variables. In addition, execute the following commands at the U-Boot prompt to prepare for NFS deployment:

=>setenv bootargs root=/dev/nfs rw nfsroot=<tftp_serverip>:<nfs_root_path> ip=<board_ipaddr>:<tftp_serverip>:<your_gatewayip>:<your_netmask>:<board_name>:eth0:off console=ttyS0,115200

=>setenv netdev <ethx>
=>saveenv

NOTE

<ethx> is the port connected on the Linux boot network.

Now U-Boot is ready for NFS deployment.

3.9.3.4 Booting up the System

TFTP the kernel image to the board, then boot it up.

=>tftp 83000000 <uImage name>
=>tftp 8f000000 <platform dtb name>
=>bootm 83000000 - 8f000000

Now the board will boot up with NFS filesystem.

4 LS1043ARDB and LS1046ARDB Platform

4.1 LS1043ARDB Platform

4.1.1 Introduction

The LS1043A reference design board (RDB) is the high-performance computing,evaluation,and development platform that supports the QorIQ LS1043A processor.

4.1.2 Building Images

After install Yocto of this release, follow these steps to do builds using Yocto Project. Be sure to set up the host environment before doing these steps:

1. *$ . ./fsl-setup-env -m ls1043ardb-32b*
2. *$ cd <sdk-install-dir>/build_ls1043ardb-32b*
3. *$ bitbake <image-target>*

   Where <image-target> is one of the following:
   - fsl-image-minimal: contains basic packages to boot up a board
   - fsl-image-core: contains common open source packages and NXP specific packages.
   - fsl-image-full: contains all packages in the full package list.
   - fsl-image-kernelitb: A FIT image comprising the Linux image, dtb and rootfs image.
   - fsl-image-mfgtool: contains all the user space apps needed to deploy the fsl-image-mfgtool image to a USB stick, hard drive, or other large physical media.
   - fsl-toolchain: the cross compiler binary package
   - package-name(xenomai): build a specific package

Then images can be found in the following directory:

<sdk-install-dir>/build_ls1043ardb-32b/tmp/deploy/images/ls1043ardb-32b

4.1.3 Reference

More information about LS1043ARDB can be found at QorIQ SDK v2.0-1611 online documentation:

https://freescale.sdlproducts.com/LiveContent/web/pub.xql?c=t&action=home&pub=QorIQ_SDK&lang=en-US

QorIQ SDK v2.0-1611 Documentation

>> Deploy U-Boot, Linux Kernel, and Root Filesystem to a Reference Design Board (RDB)

>> Supported Boards

>> LS1043ARDB

4.2 LS1046ARDB Platform

4.2.1 Introduction

The LS1046A reference design board (RDB) is the high-performance computing, evaluation, and development platform that supports the QorIQ LS1046A processor.

4.2.2 Building Images

After install Yocto of this release, follow these steps to do builds using Yocto Project. Be sure to set up the host environment before doing these steps:

1. *$ . ./fsl-setup-env -m ls1046ardb-32b*
2. *$ cd <sdk-install-dir>/build_ls1046ardb-32b*
3. *$ bitbake <image-target>*
   Where <image-target> is one of the following:
   - fsl-image-minimal: contains basic packages to boot up a board
   - fsl-image-core: contains common open source packages and NXP specific packages.
   - fsl-image-full: contains all packages in the full package list.
   - fsl-image-kernelitb: A FIT image comprising the Linux image, dtb and rootfs image.
   - fsl-image-mfgtool: contains all the user space apps needed to deploy the fsl-image-mfgtool image to a USB stick, hard drive, or other large physical media.
   - fsl-toolchain: the cross compiler binary package
   - package-name(xenomai): build a specific package

Then images can be found in the following directory:

<sdk-install-dir>/build_ls1046ardb-32b/tmp/deploy/images/ls1046ardb-32b

4.2.3 Reference

More information about LS1046ARDB can be found at QorIQ SDK v2.0-1611 online documentation:

https://freescale.sdlproducts.com/LiveContent/web/pub.xql?c=t&action=home&pub=QorIQ_SDK&lang=en-US

QorIQ SDK v2.0-1611 Documentation

>> Deploy U-Boot, Linux Kernel, and Root Filesystem to a Reference Design Board (RDB)

>> Supported Boards

>> LS1046ARDB

4.3 ARMv8 AARCH32 Mode

Some of QorIQ Processors are compatible with ARMv8 architecture, such as LayerScape platforms. The LayerScape platforms include LS1043A, LS1046A and so on.

The ARM architecture v8, ARMv8 supports two Execution states,

• A 64-bit Execution state, AArch64.

• A 32-bit Execution state, AArch32, which is compatible with previous versions of the ARM architecture.

In industry v0.1 release, LS1043ARDB and LS1046ARDB platform run in AARCH32 mode by default.

ARMv8 AArch32 User Manual can be found at QorIQ SDK v2.0-1611 online documentation:

https://freescale.sdlproducts.com/LiveContent/web/pub.xql?c=t&action=home&pub=QorIQ_SDK&lang=en-US

QorIQ SDK v2.0-1611 Documentation

>> Additional Linux Use Cases

>> ARMv8 AArch32 User Manual

5 Xenomai RT Linux

5.1 Xenomai Introduction

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 Xenomai runs over mainline desktop and server architectures as well.

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.

More information can be found at Xnomai official website: http://xenomai.org/

5.2 Xenomai Running Mode

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.

The dual kernel core is codenamed Cobalt, the native Linux implementation is called Mercury.
For Mercury mode, we can use PREEMPT-RT patch to meet stricter response time requirement than standard kernel preemption.

We only support Mercury mode in industry release v0.1.

5.3 Building Kernel Image for Xenomai Mercury Mode

Industry release v0.1 support the following platform for Xenomai Mercury mode: ls1043ardb ls1046ardb, ls1021atsn.

The following is default kernel configuration file used by each platform:

Board	defconfig
ls1021atsn	multi_v7_defconfig + freescale.config + freescale_rt.config
ls1043ardb-32b	ls_aarch32_defconfig
ls1046ardb-32b	ls_aarch32_defconfig

By default, the following kernel configuration item is enabled to use full PREEMPT RT mode:

Kernel Features
-- Preemption Model
-- (X) Fully Preemptible Kernel (RT)

5.4 Building Xenomai Mercury Tools

Xenomai libraries and tools has already built into fsl-image-core and fsl-image-kernelitb by default.

In addition, they are installed in "/usr/xenomai" directory of target board's file system.

Xenomai module can be built by the following command if they are updated:

```
bitbake xenomai
```

5.5 Running Xenomai Mercury

Xenomai Mercury provide the following API reference:

Test programs:

- latency
  Xenomai timer latency benchmark, user manual can be found at:
  http://www.xenomai.org/documentation/xenomai-3/html/man1/latency/index.html
- cyclictest
  Xenomai high resolution timer test, user manual can be found at:
  http://www.xenomai.org/documentation/xenomai-2.6/html/cyclictest/index.html

Utilities:

- xeno
  Wrapper for Xenomai executables, user manual can be found at:
  http://www.xenomai.org/documentation/xenomai-2.6/html/xeno/index.html
- xeno-config
  Display Xenomai libraries configuration, user manual can be found at:
  http://www.xenomai.org/documentation/xenomai-2.6/html/xeno-config/index.html

6 TSN Demo

6.1 Introduction

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

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

On NXP platforms, TSN features are provided by the **SJA1105TEL** Automotive Ethernet switch present on the LS1021ATSN board. The features can be used to implement the following standards:

- 802.1Qbv - Time Aware Shaping
- 802.1Qci - Per-Stream Filtering and Policing

6.2 Demo overview

The TSN demo employs 3 Linux machines connected through the SJA1105 switch. Of these 3 machines, one is the LS1021 SoC and the other 2 should be laptops or PC's connected through Ethernet cables to the LS1021ATSN board. A diagram of the connections required is shown in Figure 6-2. 1000Mbps Ethernet ports are required on the 2 GNU/Linux laptops.

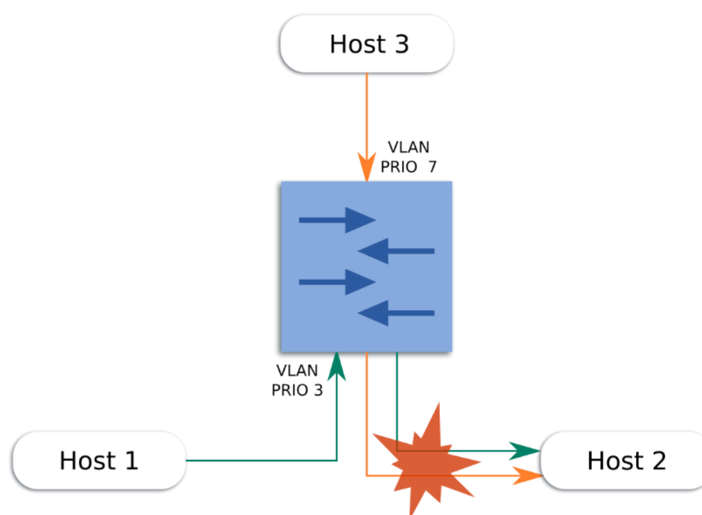

Figure 6-1 *Flows directed from Host 1 and Host 3 towards Host 2 are bottlenecked at the switch egress interface*

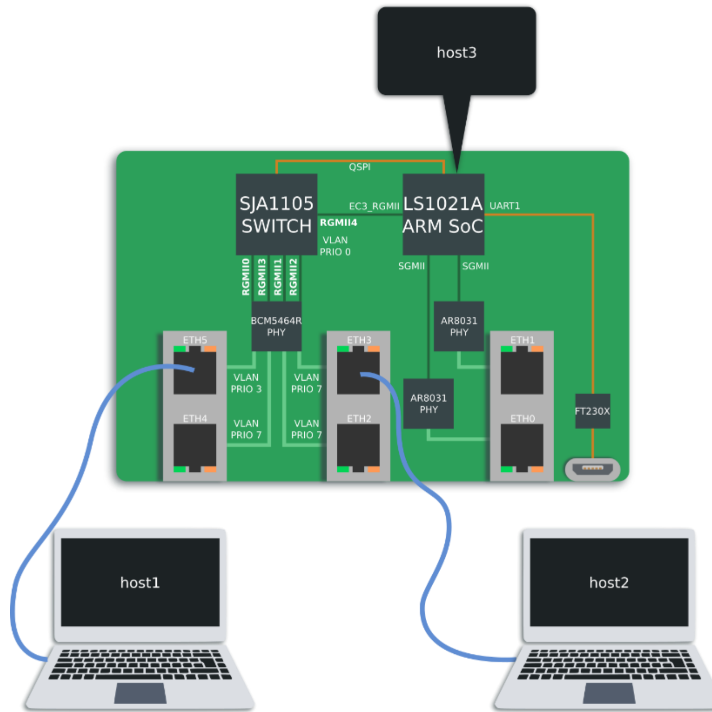

Figure 6-2 *Diagram of connections of the laptops to the board and their role in the TSN demo*

Through the SJA1105 switch there are two TCP flows competing for bandwidth:

- An iperf connection running from client Host 1 to server Host 2
- An iperf connection running from client Host 3 to server Host 2

The schematic diagram of the two flows is shown in Figure 6-1. A correlation can be drawn between Figure 6-1 and Figure 6-2. The LS1021 (Host 3) and one of the Linux laptops (Host 1) are acting as iperf clients, whereas the second Linux laptop (Host 2) is the iperf server.

Because both these flows share the same link between the SJA1105 and Host 2, they are bottlenecked and competing for the 1000Mbps total bandwidth of that link. The demo shows 3 approaches to isolate the flows' impact on one another:

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

6.3 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: this 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 3 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, each flow gets its own time slot allocated for the forwarding and egress stage.

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

To summarize:

- The switch receives untagged frames on ingress
- The switch sends untagged frames on egress
- The "native" VLAN tag is only considered during the forwarding and egress stage

The exact mapping of flows to VLAN headers is described in Figure 6-3.

- Packets coming from Host 1 are received on SJA1105 port RGMII 0 (marked ETH5 on the chassis) and are tagged with native VLAN ID 0 and PRIO 3.
- Packets coming from Host 2 are received on SJA1105 port RGMII 2 (marked ETH3 on the chassis) and are tagged with native VLAN ID 0 and PRIO 7.
- Packets coming from Host 3 are received on SJA1105 port RGMII 4 (internal connection on the board) and are tagged with native VLAN ID 0 and PRIO 0.

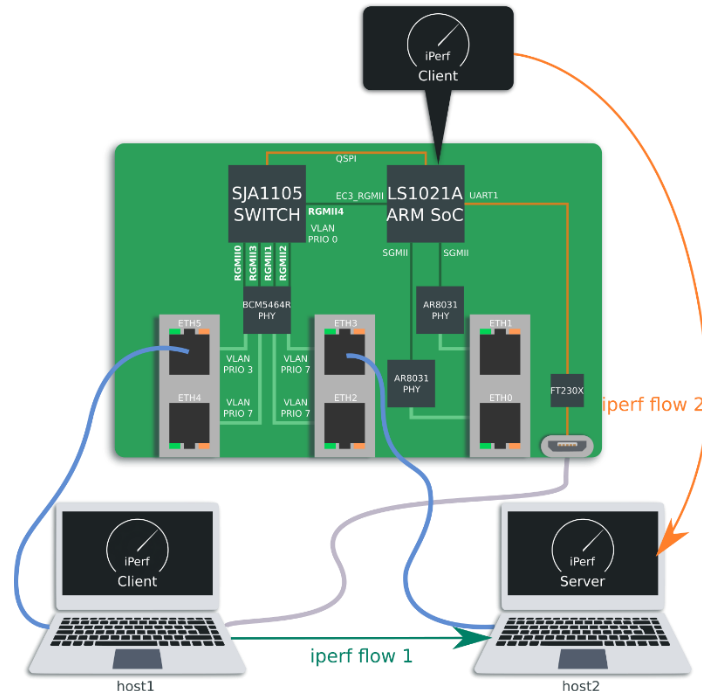

Figure 6-3 *Native VLAN tags shown for the iPerf flows*

Note that priorities 3, 7 (highest) and 0 (lowest) are assigned to hosts only in the second (policing) and third (scheduling) approach. In the standard setup, all hosts are given equal VLAN priority 0 (best effort).

6.4 Setup preparation

1. The following packages must be installed on Host 1 and Host 2 (commands are valid for Ubuntu):

   $ sudo apt-get install iperf screen rsync git openssh-client wget gnome-terminal

2. On Host 1 *extract* the TSN demo scripts and configuration files from *Industry-Linux-Solution-Release-V0.1-CORTEXA7-IMAGE-20170120-yocto.iso*. The scripts are located inside the folder "ls1021atsn-demo". Copy this folder to Host 2 as well.

3. Deploy the rootfs on a microSD card for Host 3 (the LS1021ATSN board) according to chapter 3.4.2 Target Board Startup and 3.8.3 SD Deployment. Then to install the configuration files specific for the TSN demo run:

   $ cd ls1021atsn-demo
   $ sudo ./setup/scripts/deploy-demo --local /media/user/rfs/

   In the example above, the microSD card is assumed to be mounted at the /media/user/rfs location.

4. Boot the LS1021ATSN board with the microSD rootfs updated to include the TSN configuration files. These set up a SSH server listening on the LS1021 interface (eth2) connected through the SJA1105 switch to chassis ports ETH2, ETH3, ETH4 and ETH5. Once booted, this interface of the LS1021 has a static IP of 172.15.0.1.

5. Connect to Host 3 to set up DHCP server:

   $ ssh root@172.15.0.1 # no password

   $ /etc/init.d/dhcp-server start #Set up DHCP server on Host 3

   Connect Ethernet cables from Host 1 and Host 2 to the LS1021ATSN board as described in Figure 1. This means connecting Host 1 to Ethernet port ETH5 and Host 2 to ETH3. These go to the SJA1105 ports RGMII0 and RGMII2.Wait until the hosts receive an IP through DHCP.

6. If the step above fails, use a microUSB cable to connect from Host 1 to Host 3 through a serial interface. You can use the following debugging tips:

```
$ screen /dev/ttyACM0 115200          # to connect to Host 3 from Host 1
$ /etc/init.d/dhcp-server start       # if the DHCP server on Host 3 stopped
$ cat /var/log/dhcpd.log              # read DHCP server logs on Host 3
$ sja1105-tool config upload          # reload the current switch config on Host 3
$ get-general-status.sh        # see if the TSN switch is properly configured on Host 3
$ get-port-status.sh                  # see if packets are dropped by the switch on Host 3
$ sudo dhclient –v eth0        # run this on Host 1 to inspect DHCP process  manually
$ tcpdump -i eth2 -vnes0 "icmp or arp or port 67 or port 68"
                                      # run on Host 3 to inspect ping and DHCP traffic
$ sudo arp -s 172.15.0.1 "<mac-of-eth2-on-ls1021>" dev <eth-port>
                                      # on Host 1 you may add a static ARP entry
                                      # towards Host 3 for debugging
$ /etc/init.d/sshd stop        # Apply these steps on Host 3
$ rm -f /etc/ssh/ssh_host_*_key*      # (LS1021) if you have issues
$ ssh-keygen –A                       # with the SSH server
$ /etc/init.d/sshd start       # or keys
```

$ ssh-keygen -f "~/.ssh/known_hosts" -R 172.15.0.1

 # Needs to be run afterwards on the SSH

 # client (Host 1 or Host 2)

7. The hosts know one another's IP through Avahi mDNS
   - Host 1 advertises its name as "host1.local"
   - Host 2 advertises itself as "host2.local"
   - Host 3 has the fixed address 172.15.0.1, since it is the DHCP server

   On Host 2, advertise the "host2.local" name through Avahi mDNS and open a new iPerf server waiting for TCP connections:

   $ cd ls1021atsn-demo

   $ ./demo-host/host2/run-all   # Run on Host 2

8. On Host 1, advertise the "host1.local" name through Avahi mDNS and start an iPerf client which connects to the server on Host 2 and begins transferring bulk data at line rate:

   $ cd ls1021atsn-demo

   $ ./demo-host/host1/run-all   # Run on Host 1

9. Commands on Host 3 (LS1021) can be run remotely from either Host 1 or Host 2. These are executed through SSH but will display the output in a terminal window locally. This command starts the second iPerf client from Host 3 to Host 2:

   $ cd ls1021atsn-demo

   $ ./demo-host/host3/iperf-to-host2   # Run on Host 1 or 2, NOT on 3!

   Note: if the iPerf monitoring terminal cannot pop up, try the following to attempt debugging:

   # ./demo-host/scripts/trace ./demo-host/host3/iperf-to-host2   iperf-to-host2

6.5 SJA1105 Switch Configurations

These are stored as XML files on Host 3 (LS1021) and loaded through SPI with the help of the sja1105-tool:

- /etc/sja1105/standard-config.xml
- /etc/sja1105/policing-config.xml
- /etc/sja1105/scheduling-config.xml

For a complete description of the sja1105-tool functionality, consult the manpages:

> *$ man 1 sja1105-tool*
> *$ man 1 sja1105-tool-status*
> *$ man 1 sja1105-tool-reset*
> *$ man 1 sja1105-tool-config*
> *$ man 5 sja1105-tool-config-format*
> *$ man 5 sja1105-conf*

On Host 3 there are the following helper scripts available:

$ next-config.sh

> *# iterate through /etc/sja1105/{standard | policing | scheduling}.xml*
>
> *# and load the XML configuration into /lib/firmware/sja1105/sja1105.bin*

$ sja1105-tool config upload

> *# reload the current configuration on the switch*

$ get-general-status.sh

> *# inspect switch configuration status*

$ get-port-status.sh

> *# inspect frame statistics per port*

6.6 Analysis

6.6.1 Standard configuration

In this approach, only basic configuration is done, so no TSN features are active. The Ingress Policer is “disabled” on all ports, as well as the Time Aware Scheduler. The SJA1105 internal VLAN assignment is PRIO 0 to all ingress traffic. As such, packet forwarding is “best effort” and no flow has priority over the other.

When running the iPerf flows individually, the TCP congestion control algorithm makes them occupy as much bandwidth as possible, but when running the flows simultaneously, they will get an unfair allocation of the total 1000Mbps link to Host 2.

The bandwidths, as measured and reported by iPerf, will oscillate (and possibly reverse) over time, but their sum can never exceed 1000Mbps, the capacity of the link.

Bandwidth measurements taken with the iPerf streams running separately (individually) can be seen in Figure 6-4 and Figure 6-5, and measurements with both streams running simultaneously can be seen in Figure 6-6.

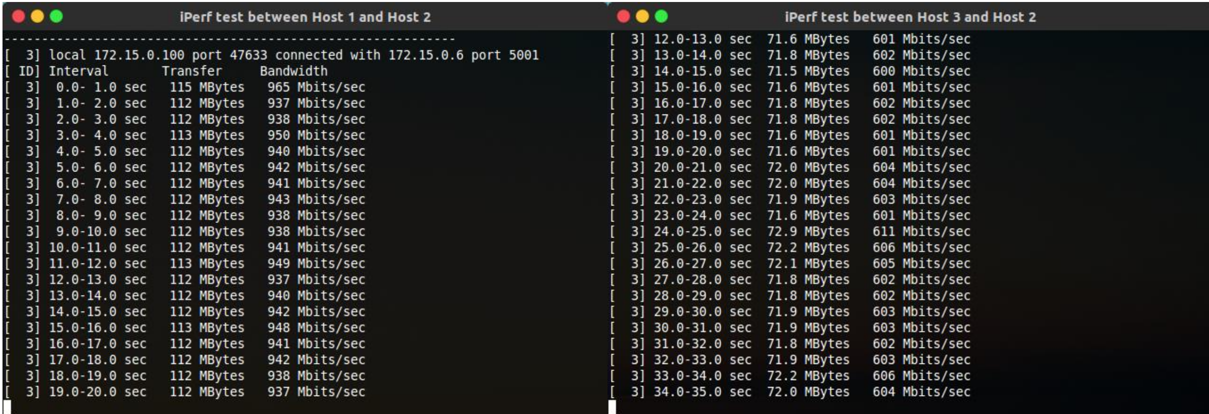

Figure 6-4 Performance of iPerf flow from Host 1 to Host 2 running on its own

Figure 6-5 Performance of iPerf flow from Host3 to Host 2 running on its own

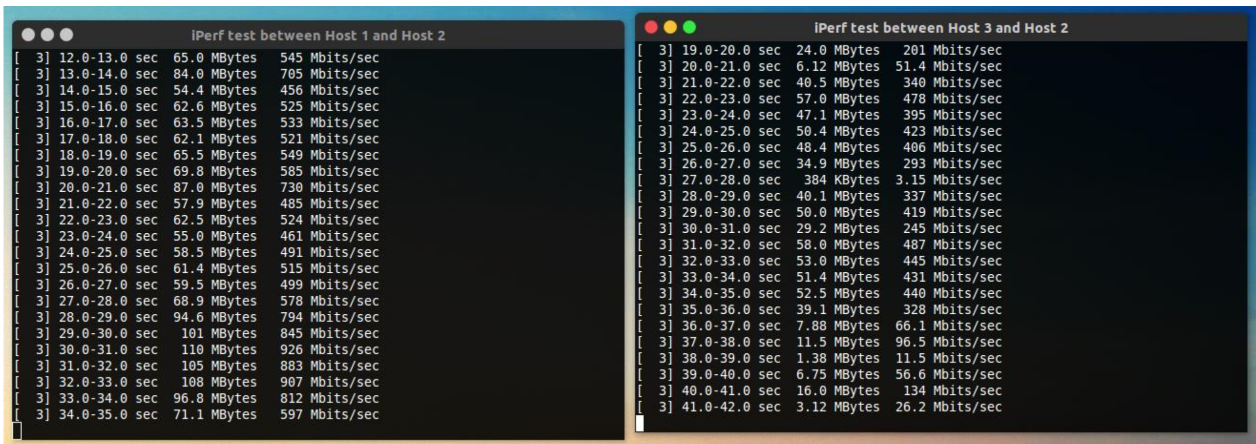

Figure 6-6 Performance of both iPerf flows running simultaneously

Packet drops can be seen by running sja1105-tool status port on Host 3 and inspecting N_PART_DROP for insufficient buffer memory on the switch, and N_QFULL for tail drops.

6.6.2 Policing configuration

The Ingress Policer of the SJA1105 is loosely related to the 802.1Qci (Per-stream Filtering and Policing) draft specification. The standard talks about detecting misbehaving (“babbling”) streams and blocking them, which can be done.

The Policer inside SJA1105 is implemented as a Token Bucket Shaper: the bucket size (maximum burst size) is called nSMax and the rucket refill speed is nRate bytes per second (up to a maximum of 64000).

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)

The Ingress Policer can also drop ingress packets based on maximum frame size and other criteria.

In the second approach iPerf flow 2 (from Host 3 to Host 2) is treated as non-critical and as such, the Ingress Policer is configured to rate-limit its traffic:

- Ingress Port 0, VLAN PRIO 3 (Host 1) – 1000Mbps
- Ingress Port 2, VLAN PRIO 7 (Host 2) – 1000Mbps
- Ingress Port 4, VLAN PRIO 0 (Host 3) – 400Mbps

Note that the value configured in the Policing table is not a hard guarantee: Host 3 cannot exceed 400Mbps, but it doesn't mean it can always achieve it, so flow 2 might suffer starvation in this case.

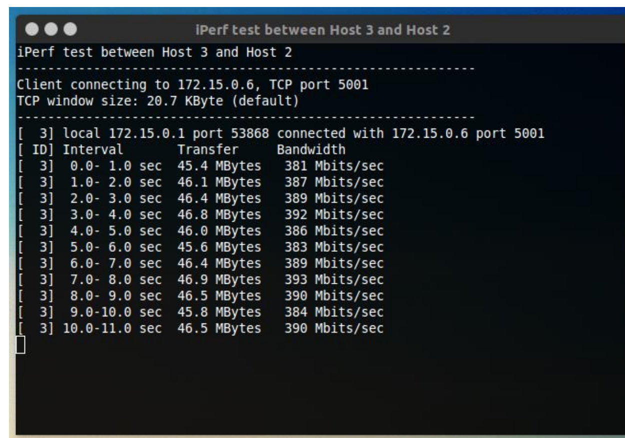

Figure 6-7 *Performance of iPerf flow from Host 3 to Host 2 running on its own. Because of the Ingress Policer, this is limited to 400Mbps*

Bandwidth measurements taken with the iPerf stream from Host 3 running individually (and rate-limited to 400Mbps) can be seen in Figure 6-7 and measurements with both streams running simultaneously can be seen in Figure 6-8.

The packet drops caused by the Ingress Policer's action can be seen by running the get-port-status.sh command and inspecting N_POLERR.

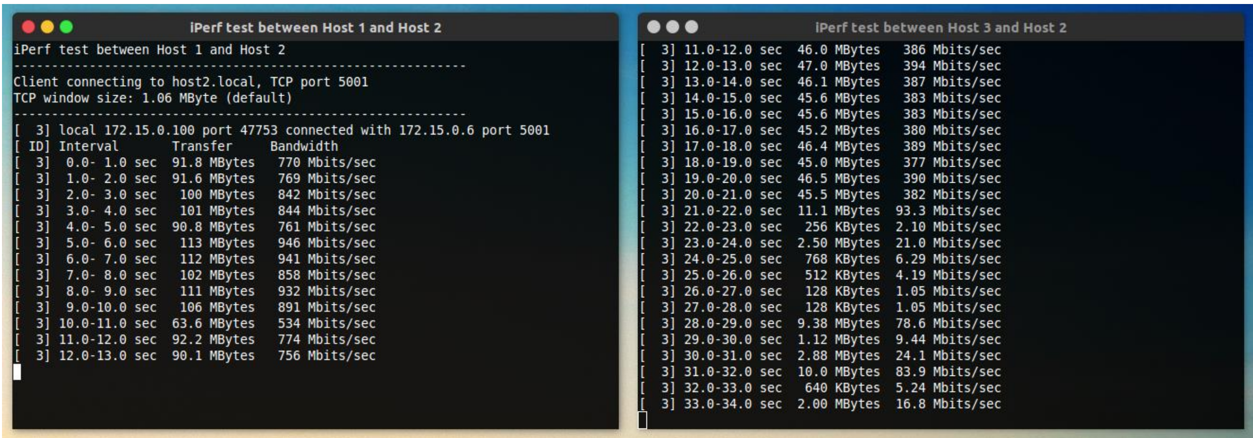

Figure 6-8 Performance of both iPerf flows running simultaneously. Host 3 suffers from starvation

6.6.3 Scheduling configuration

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

- The 5 Egress Ports each have 8 Gates, which can be open or closed
- Each Gate has 1 Queue associated with it
- Whenever a Gate is open, packets from that Queue can be sent out the wire
- An internal clock generates ticks each 200ns
- At each tick, a new time slot can be created, where some Gates can be opened and some can be closed

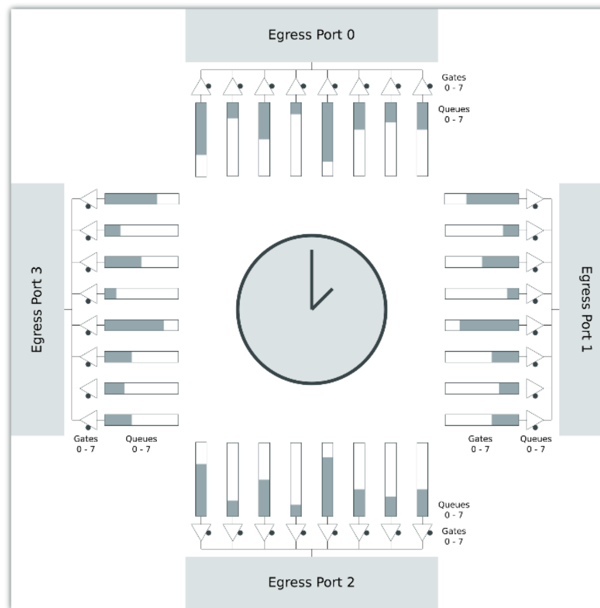

Figure 6-9 Structure of the Time Aware Scheduler

The user defines how many clock ticks each time slot (called subschedule) 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 the third approach of the demo, the Time Aware Scheduler is active for Egress Port 2. This is the link towards Host 2, where the contention between Flow 1 and Flow 2 happens. The SJA1105 switch is configured to create a subschedule for VLAN PRIO 0 and one for PRIO 3, each having an equal amount of time. This way, Flow 1 is completely isolated from Flow 2, and there is minimal interference between, the two, which allows the best utilization of bandwidth.

In **Figure 6-10**, Flow 1 and Flow 2 are depicted with the same colors as in **Figure 6-2**. Flow 1 has packets coming from Host 1, so they are tagged with VLAN PRIO 3. Flow 2 has packets coming from Host 3, so they are tagged with VLAN PRIO 0.

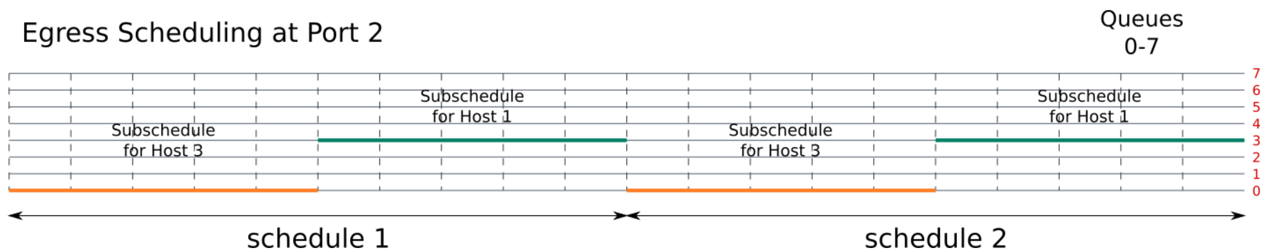

Figure 6-10

Bandwidth measurements taken with the iPerf streams running simultaneously from Host 1 and Host 3 can be seen in Figure 6-11. Because the subschedules are of equal durations, both flows reach almost half of the total maximum bandwidth of the 1000Mbps interface.

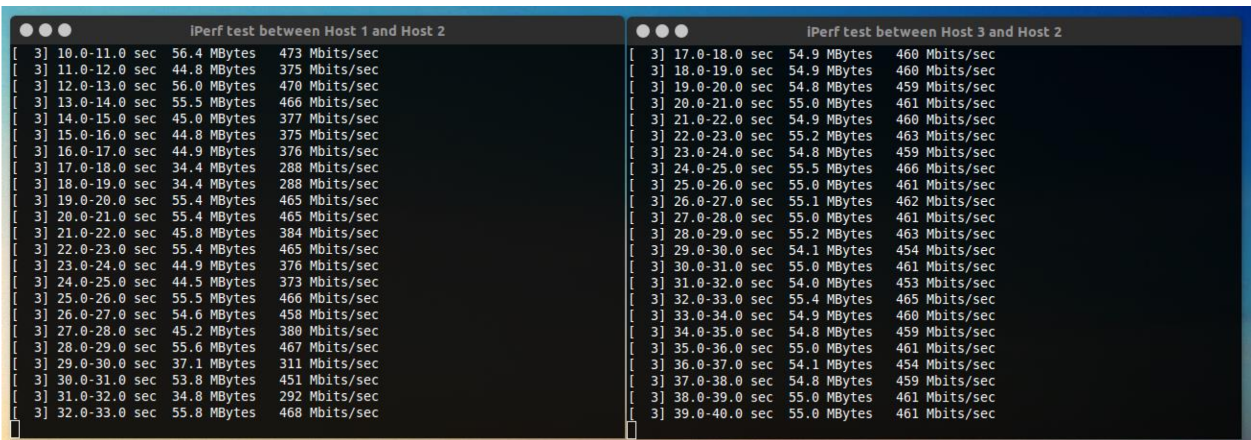

Figure 6-11 Performance of both iPerf flows running simultaneously

7 IEEE 1588

7.1 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, PTPd which are used for this release demo.

7.2 PTP device types

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

a) 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, i.e., be a master clock, or may synchronize to another clock, i.e., be a slave clock.

b) 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 the source of time, i.e., be a master clock, and may synchronize to another clock, i.e., be a slave clock.

c) 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.

d) 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-topeer

delay measurement mechanism.

e) Management node

A device that configures and monitors clocks.

Note 1: 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.

7.3 linuxptp stack

7.3.1 Introduction

This software is an implementation of the Precision Time Protocol (PTP) according to IEEE standard 1588 for Linux. The dual design goals are to provide a robust implementation of the standard and to use the most relevant and modern Application Programming Interfaces (API) offered by the Linux kernel. Supporting legacy APIs and other platforms is not a goal.

7.3.2 Features

- 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 allowing painless addition of new transports and clock servos.

7.4 PTPd stack

7.4.1 Introduction

The PTP daemon (PTPd) implements the Precision Time protocol (PTP) as defined by the IEEE 1588 standard. PTP was developed to provide very precise time coordination of LAN connected computers.

7.4.2 Features

- Full IEEE 1588-2008 (PTPv2) protocol implementation
- Software-only timestamping, advanced filtering, robust to network failures
- Multi-platform: Linux, FreeBSD, OpenBSD, NetBSD, Solaris, various embedded OS
- Support for PTP Default profile, Telecom profile (G.8265) and Enterprise profile
- IPv4 Multicast, unicast and Ethernet operation
- Highly configurable
- Advanced Leap Second handling
- Integration with NTPd as failover time source

7.5 Quick start guide

7.5.1 Requirement

Hardware requirement

- Two boards for basic master-slave synchronization
- Three or more boards for BC synchronization

Software requirement

- Linux BSP of Industry Solution Release
- PTP software stack

Note1: PTPd stack only supports basic master-slave synchronization

7.5.2 Ethernet interfaces connection

Basic master-slave synchronization

Connect two Ethernet interfaces between two boards in back-to-back way. Then one board will be master and the other will be slave when they synchronize. Both the master and the slave are working as OCs.

BC synchronization

Three boards are required at least for BC synchronization. When three boards are used for BC synchronization. Assume board A will work as BC with two PTP ports. Board B and board C works as OCs.

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

1. Connect board A interface 1 to board B interface 1 in back-to-back way.
2. Connect board A interface 2 to board C interface 1 in back-to-back way.

For example, LS1021ATSN BC synchronization connection is as Figure 7-1.

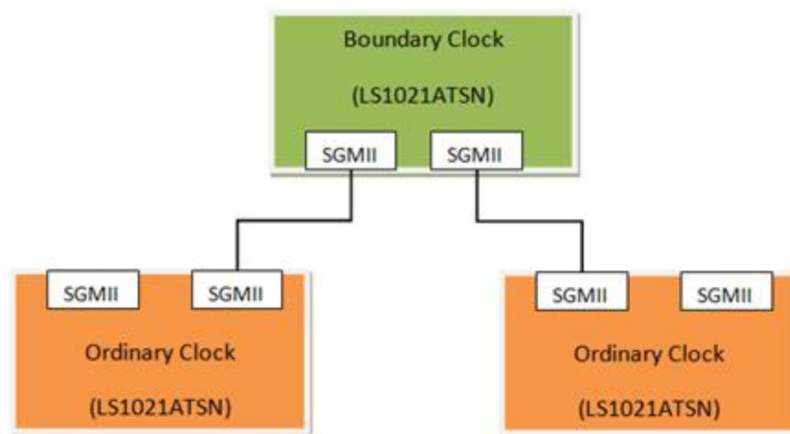

Figure 7-1 LS1021ATSN BC test setup

7.5.3 PTP stack startup

Before starting up 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*

PTPd stack

PTPd stack only supports ordinary clock. It is included in full rootfs image of LS1043ARDB and LS1046ARDB (like this fsl-image-full-<board>-<release date>-rootfs.tar.gz). If we want to compile this stack in yocto alone, use below command to generate the binary file “ptpd2”, then copy this file to the target board.

$ bitbake ptpd

Run below command for master-slave synchronization.

Master

$ ptpd2 -i eth0 -MV

Slave

$ ptpd2 -i eth0 -sV --servo:kp=0.32 --servo:ki=0.05

Note: The stack uses default /dev/ptp0 device. If there are more than 1 ptp clocks, please use '-o' argument to clarify the ptp clock of DPAA we use. For example,

$ ptpd2 -i eth0 -MV -o /dev/ptp1

Linuxptp stack

Linuxptp stack supports both OC and BC. It is included in below images of LS1021ATSN:

- Full rootfs image (like this fsl-image-full-<board>-<release date>-rootfs.tar.gz)
- SD card image (like this ls1021atsn_sd.img)

If we want to compile this stack in yocto alone, use below command to generate the binary file "ptp4l", then copy this file to the target board.

$ bitbake linuxptp

Below command is used for running it. The master could be selected by software BMC (Best Master Clock) algorithm.

$ ptp4l -i eth0 -p /dev/ptp0 -f default.cfg -m

If the board is used as BC with several PTP ports, the '-i ' argument could point more interfaces. Such as below command for two PTP ports BC.

$ ptp4l -i eth0 -i eth1 -p /dev/ptp0 -f default.cfg -m

The default.cfg could be used or not, it's convenient to use it to change configuration. The configuration 'logSyncInterval' and 'summary_interval' could be changed to -3 from default 0 to get faster time offset convergence.

Note: The interface name and PTP device name in commands should be changed accordingly.

7.6 Known issues and limitations

See item3 and item4 in Chapter 8: Known Issues.

7.7 Long term test results

Linuxptp

Connection: back-to-back master to slave

Configuration: Sync internal –3

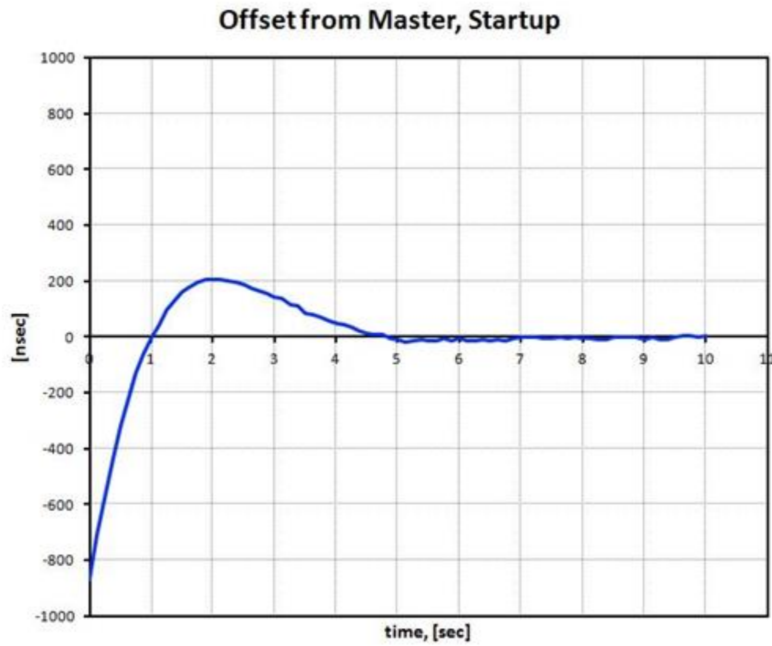

Figure 7-2 Offset from Master at startup

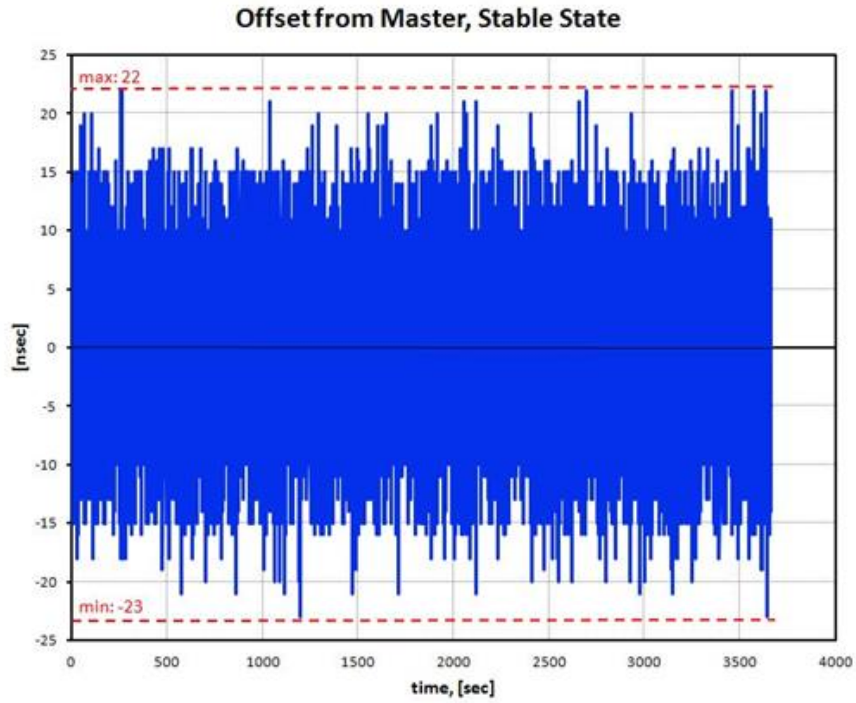

Figure 7-3 Offset from Master at stable state

8 Known Issues

Following is the list of known issues in this release:

CQ #	Description
1	Kernel tracer is not supported on LS1043ARDB and LS1046ARDB AARCH32 platform, kernel will hang when enable “CONFIG_FTRACE” item. **Workaround**: Do not enable “CONFIG_FTRACE”.
2	Unable to utilize full bandwidth of eTSEC Ethernet controller on LS1021ATSN board.
3	The linuxptp stack only supports LS1021A Ethernet interfaces. It couldn’t be used for SJA1105 switch Ethernet interfaces. **Workaround:** Use LS1021A Ethernet ports instead of SJA1105 switch Ethernet ports for IEEE1588 testing.
4	Packet loss issue could be observed on LS1021ATSN SGMII interfaces connected in back-to-back way. The root cause is that the PHY supports IEEE 802.11az EEE mode by default. The low speed traffic will make it go into low power mode. It affects 1588 synchronization performance greatly. **Workaround:** disable IEEE 802.11az EEE mode. *$ ifconfig eth0 up* *$ ethtool --set-eee eth0 advertise 0* *$ ifconfig eth0 down* *$ ifconfig eth0 up*

Appendix A. Version Tracking

Date	Version	Comments	Author(s)
22/11/2016	0.1	Draft Version	Jiafei Pan
16/12/2016	0.2	Update Architecture, added Xenomai draft	Jiafei Pan
16/12/2016	0.3	Added LS1021ATSN and LS1021AIOT platform document	Xiaoliang Yang
23/12/2016	0.4	Added “Getting Start” chapter	Jiafei Pan
23/12/2016	0.5	Added “IEEE 1588” chapter	Yangbo Lu
23/12/2016	0.6	Added “OpenWrt” chapter	Liu Gang
26/12/2016	0.7	Split TSN and IOT into two document	Jiafei Pan
28/12/2016	0.8	Added “TSN Demo” chapter	Vladimir
19/01/2017	0.9	Flash RCW	Xiaoliang Yang
22/01/2017	1.0	Change known issue list	Mingkai Hu