RTEDGEYOCTOUG

Real-time Edge Yocto Project User Guide

Rev. 3.2 — 29 July 2025 **User guide**

Document information

Information	Content
Keywords	RTEDGEYOCTOUG, Real-time Edge software Yocto layer, i.MX boards, Layerscape boards, Yocto project setup, image building, boot options, eMMC, Real-time networking recipes
Abstract	This document describes Real-time Edge software Yocto layer and its usage. It includes steps to build a Real-time Edge image for both i.MX and Layerscape boards by using a Yocto project build environment.

1 Overview

This document describes how to build a Real-time Edge image for both i.MX and QorIQ (Layerscape) boards by using a Yocto Project build environment. It describes Real-time Edge Software Yocto layer and its usage.

The Yocto Project is an open source collaboration that focuses on embedded Linux OS development.

- For more information on Yocto Project, see the Yocto Project page: www.yoctoproject.org. There are several documents on the Yocto Project homepage that describe in detail how to use the system.
- To use the basic Yocto Project without the Real-time Edge release layer, follow the instructions in the Yocto Project Quick Start found at https://docs.yoctoproject.org/brief-yoctoprojectqs/index.html.

Real-time Edge layer is based on i.MX Yocto project and LSDK Yocto release.

- i.MX Yocto project provides i.MX boards support. For more information, refer to *i.MX Yocto Project User's Guide (UG10164)*.
- LSDK Yocto project provides Layerscape boards support. For more information, refer to *Layerscape Software Development Kit User Guide for Yocto (LSDKYOCTOUG)*.

Files used to build an image are stored in layers. Layers contain different types of customizations and may have different sources. Some of the files in a layer are called recipes. Yocto Project recipes contain the mechanism to retrieve source code, build, and package a component. The following list shows the layers used in this release.

Real-time Edge layer

- **dynamic-layers**: includes updates for board-related recipes of i.MX and Layerscape.

```
├── imx-layer
└── qoriq-layer
```

- **recipes-extended**: includes recipes for real-time networking, real-time system, and industrial protocols.
- **recipes-nxp**: Real-time Edge image recipes

1.1 End user license agreement

NXP End-User License Agreement (EULA) is displayed during the setup process of the Real-time Edge Yocto Project Community Board Support Package (BSP). To continue using the Real-time Edge Proprietary software, users must agree to the conditions of this license. The agreement to the terms allows the Yocto Project build to untar the packages.

1.2 Related documentation

- For more information about i.MX Yocto project, refer to *i.MX Yocto Project User's Guide* (UG10164).
- For more information about LSDK Yocto project, refer to *Layerscape Software Development Kit User Guide for Yocto* (*LSDKYOCTOUG*).
- For more information about the real-time features, refer to the *Real-time Edge Software User Guide* on https://www.nxp.com/design/software/development-software/real-time-edge-software:REALTIME-EDGE-SOFTWARE?tab=Documentation_Tab.
- For details of the new features and release information, refer to the Real-time Edge Release Notes (RN00161)
- For detailed instructions on booting up and setting up the relevant boards, refer to the User Guide of the respective board.

2 Features

Real-time Edge Yocto Project has the following features:

- **Linux kernel recipe**
  - The kernel recipe contains two folders:
    - `dynamic-layers/imx-layer/recipes-kernel`: Linux for i.MX boards
    - `dynamic-layers/qoriq-layer/recipes-kernel`: Linux for Layerscape boards
  - `Linux 6.12.20` is the base of Linux kernel released for the Yocto Project.
- **U-Boot recipe**
  - The U-Boot recipe has two folders:
    - `dynamic-layers/imx-layer/recipes-bsp/u-boot/`: U-Boot for i.MX boards
    - `dynamic-layers/qoriq-layer/recipes-bsp/u-boot/`: U-Boot for Layerscape boards
  - `U-Boot 2025.04` is the U-Boot base released for the Yocto Project.
  - `u-boot-script-distroboot` recipe provides distro bootscript for normal and BareMetal images.
- **Real-time Networking recipes**
  - `avahi`
  - `iproute2`
  - `genavb-tsn`
  - `libredblack`
  - `libyang`
  - `lldpd`
  - `linuxptp`
  - `netopeer2-cli`
  - `netopeer2-keystored`
  - `netopeer2-server`
  - `real-time-edge-sysrepo`
  - `sysrepo`
  - `tsn-scripts`
  - `tsntool`
- **Real-time System recipes**
  - `real-time-edge-baremetal`: Real-time Edge BareMetal recipe resides in `recipes-extended` directory. It provides BareMetal binary run on responder cores.
  - `real-time-edge-icc`: The 'icc' recipe resides in `recipes-extended` directory. It provides a tool to community between master/slave and slave/slave cores.
  - `jailhouse`: The 'Jailhouse' recipe resides in `recipes-extended` directory. It is a partitioning hypervisor based on Linux.
- **Protocols recipes**
  - `igh-ethercat`
  - `libnfc-nci`
  - `libopen62541`
  - `real-time-edge-libbee`
  - `real-time-edge-libblep`
  - `real-time-edge-servo`
- **Harpoon recipes**
  Harpoon recipes reside in the meta-layer `meta-nxp-harpoon`.
  - `harpoon-apps-freertos-audio, harpoon-apps-zephyr-audio`: directory `recipes-bsp/harpoon-apps`. It provides the Harpoon audio applications running on inmate cell of Jailhouse.

  - harpoon-apps-freertos-rt-latency, harpoon-apps-zephyr-rt-latency: directory recipes-bsp/harpoon-apps. It provides a latency test application running on inmate cell of Jailhouse (running on FreeRTOS or Zephyr).
  - harpoon-apps-ctrl: directory recipes-bsp/harpoon-apps. It provides a control application running on Linux side to communicate with the inmate cell of Harpoon Jailhouse. It also provides helper scripts to start and stop the inmate cell of Harpoon Jailhouse.
  - harpoon-apps-freertos-industrial, harpoon-apps-zephyr-industrial: directory recipes-bsp/harpoon-apps. It provides the Harpoon industrial applications running on inmate cell of Jailhouse (running on FreeRTOS or Zephyr).
- **AVB endpoint recipes**
  AVB endpoint recipes reside in meta-layer meta-nxp-avb.
  - genavb-tsn and genavb-media: directory recipes-avb. It provides the recipes to build and install AVB endpoint stack binaries, demo media applications, and media files.

3 Host setup

To get the desired Yocto Project behavior in a Linux Host Machine, the packages and utilities described in the following sections must be installed. An important consideration is the hard disk space required in the host machine. For example, while building on a machine running Ubuntu, the minimum hard disk space required is about 50 GB. It is recommended that at least 120 GB is provided, which is enough to compile all backends together. For building machine learning components, at least 250 GB is recommended.

The recommended minimum Ubuntu version is 20.04 or later. The latest release supports Chromium v91, which requires an increase to the `ulimit` (number of open files) to 4098.

3.1 Host packages

To build a Yocto Project, few packages must be installed. These packages are documented under the Yocto Project. Go to https://docs.yoctoproject.org/brief-yoctoprojectqs/index.html and check for the packages that must be installed for your build machine.

The essential Yocto Project host packages are:

```
$ sudo apt install gawk wget git diffstat unzip texinfo gcc build-essential \
chrpath socat cpio python3 python3-pip python3-pexpect xz-utils debianutils \
iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-dev \
python3-subunit mesa-common-dev zstd liblz4-tool file locales -y
$ sudo locale-gen en_US.UTF-8
```

Attention: *The configuration tool uses the default version of `grep` that is on your build machine. If there is a different version of `grep` in your path, it might cause builds to fail. One workaround is to rename the special version to something not containing the term '`grep`'.*

3.2 Setting up the repo utility

'Repo' is a tool based on Git that makes it easier to manage projects containing multiple repositories, provided they do not need to be on the same server. Repo complements very well the layered nature of the Yocto Project, making it easier for users to add their own layers to the Board Support Package (BSP).

To install the 'repo' utility, perform these steps:

1. Create a `bin` folder in the home directory.

```
$ mkdir ~/bin (this step may not be needed if the bin folder already exists)
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

2. Add the following line to the `.bashrc` file to ensure that the `~/bin` folder is in your `PATH` variable.

```
$ export PATH=~/bin:$PATH
```

4 Yocto Project setup

First, make sure that Git is set up properly using the commands below:

```
$ git config --global user.name "Your Name"
$ git config --global user.email "Your Email"
$ git config --list
```

The Real-time Edge Yocto Project Release directory contains a `sources` directory. This directory contains the recipes used to build one or more build directories, and a set of scripts used to set up the environment.

The recipes used to build the project come from both the community and Real-time Edge. The Yocto Project layers are downloaded to the `sources` directory. In this directory, the recipes that are used to build the project are set up.

The following example shows how to download the Real-time Edge recipe layers. For this example, a directory called `yocto-real-time-edge` is created for the project. Any other name can also be used, instead of this name.

```
$ mkdir yocto-real-time-edge
$ cd yocto-real-time-edge
$ repo init -u https://github.com/nxp-real-time-edge-sw/yocto-real-time-edge.git \
-b real-time-edge-walnascar \
-m real-time-edge-3.2.0.xml
```

When this process is completed, the source code is checked out into the directory `yocto-real-time-edge/sources`. You can perform repo synchronization, with the command `repo sync`, periodically to update to the latest code.

If errors occur during repository initialization, try deleting the `.repo` directory and running the repo initialization command again.

The `repo init` is configured for the latest patches in the line.

5 Image building

This section provides the detailed information along with the process for building an image.

5.1 Build configurations

Real-time Edge provides the script `real-time-edge-setup-env.sh`, which simplifies the setup for both i.MX and Layerscape boards. To use the script, the name of the specific machine to be built for and the desired distro must be specified. The script sets up a directory and the configuration files for the specified machine and distro.

Real-time Edge supports the following NXP hardware platforms:

- `imx6ull14x14evk`
- `imx8dxlb0-lpddr4-evk`
- `imx8mm-lpddr4-evk`
- `imx8mp-lpddr4-evk`
- `imx93evk`
- `imx93-9x9-lpddr4-qsb`
- `imx93-14x14-lpddr4x-evk`
- `imx91-11x11-lpddr4-evk`
- `imx91-9x9-lpddr4-qsb`
- `imx95-19x19-lpddr5-evk`
- `imx95-15x15-lpddr4x-evk`
- `imx943-19x19-lpddr5-evk`
- `imx943-19x19-lpddr4-evk`
- `ls1028ardb`
- `ls1043ardb`
- `ls1046ardb`
- `lx2160ardb-rev2`

Each build folder must be configured in such a way that it uses only one distro. Each time the variable `DISTRO_FEATURES` is changed, a clean build folder is needed. Distro configurations are saved in the `local.conf` file in the `DISTRO` setting and are displayed when the `bitbake` command is run. Here is the list of `DISTRO` configurations:

- `nxp-real-time-edge`: This is the normal image including Real-time and industrial package without BareMetal support.
- `nxp-real-time-edge-baremetal`: This is the BareMetal image (some boards do not support this distro).
- `nxp-real-time-edge-emmc`: This is the normal image to be deployed in eMMC device (for ls1028ardb and ls1046ardb only).
- `nxp-real-time-edge-plc`: This is the specific image for PLC, with the higher performance.

The syntax for the `real-time-edge-setup-env.sh` script is shown below:

```
$ DISTRO=<distro name> MACHINE=<machine name> source real-time-edge-setup-env.sh
 -b <build dir>
```

- `DISTRO`=<distro configuration name> is the distro that configures the build environment and it is stored in: `meta-real-time-edge/conf/distro`
- `MACHINE`=<machine configuration name> is the machine name that points to the configuration file in `conf/machine` in `meta-freescale` and `meta-imx`.

- `-b <build dir>` specifies the name of the build directory created by the `real-time-edge-setup-env.sh` script.

When the script is run, it prompts the user to accept the End User License Agreement (EULA). Once the EULA is accepted, the acceptance is stored in the `local.conf` file within each build folder. The EULA acceptance query is no longer displayed for that build folder.

After the script runs, it creates a working directory that was specified using the `-b` option. A `conf` folder is created, which contains the `bblayers.conf` and `local.conf` files.

The `<build dir>/conf/bblayers.conf` file contains all the metalayers used in the Real-time Edge Yocto Project release. The `local.conf` file contains the machine and distro specifications:

```
MACHINE ??= 'imx8mp-lpddr4-evk'
DISTRO ?= 'nxp-real-time-edge'
ACCEPT_FSL_EULA = "1"
```

The `MACHINE` configuration can be changed by editing this file, if necessary.

`ACCEPT_FSL_EULA` in the `local.conf` file indicates that you have accepted the conditions of the EULA.

5.2 Choosing a Real-time Edge Yocto project image

The Yocto Project provides images that are available on different layers.

Real-time Edge provides `nxp-image-real-time-edge` image, which contains Real-time Networking, Real-time System, Protocols, and Harpoon packages.

5.3 Building an image

The Yocto Project build uses the `bitbake` command. For example, `bitbake <component>` builds the named component. Each component build has multiple tasks, such as fetching, configuration, compilation, packaging, and deploying to the target rootfs. The `bitbake` image build gathers all the components required by the image and builds in the order of the dependency per task. The first build is the toolchain along with the tools required for the components to build.

The following command is an example of how to build an image:

```
$ bitbake nxp-image-real-time-edge
```

5.4 Bitbake options

The `bitbake` command can be used to build an image by specifying the image as shown below:

```
bitbake <image name>
```

Additional parameters can be used for specific activities described below. **Bitbake** provides various useful options for developing a single component. Use the command below to run `bitbake` with a parameter:

```
$ bitbake <parameter> <component>
```

In the preceding command, `<component>` is a desired build package.

The Table 1 describes some `bitbake` options.

Table 1. Bitbake options

`Bitbake` parameter	Description
`-c fetch`	Fetches if the downloads state is not marked as done.
`-c cleanall`	Cleans the entire component build directory. All the changes in the build directory are lost. The rootfs and state of the component are also cleared. The component is also removed from the download directory.
`-c deploy`	Deploys an image or component to the rootfs.
`-k`	Continues building components even if a build break occurs.
`-c compile -f`	It is not recommended to change the source code under the temporary directory. However, if it is changed, the Yocto Project might not rebuild it unless this option is used. Use this option to force a recompile after the image is deployed.
`-g`	Lists a dependency tree for an image or component.
`-DDD`	Turns on debug 3 levels deep. Each D adds another level of debug.

5.5 Build scenarios

The following are the build setup scenarios for various configurations.

Set up the manifest and populate the Yocto Project layer sources using the commands below:

```
$ mkdir yocto-real-time-edge
$ cd yocto-real-time-edge
$ repo init -u https://github.com/nxp-real-time-edge-sw/\
yocto-real-time-edge.git \
-b real-time-edge-walnascar \
-m real-time-edge-3.2.0.xml
$ repo sync
```

The following sections give some specific examples. Replace the machine names and the backends specified to customize the commands.

5.5.1 Real-time Edge image on i.MX 8M Plus EVK

The command below builds a multimedia image with Real-time Edge packages:

```
$ DISTRO=nxp-real-time-edge MACHINE=imx8mp-lpddr4-evk \
source real-time-edge-setup-env.sh \
-b build-imx-real-time-edge
$ bitbake nxp-image-real-time-edge
```

5.5.2 Real-time Edge BareMetal image on i.MX 8M Plus EVK

The command below builds a multimedia image using BareMetal packages:

```
$ DISTRO=nxp-real-time-edge-baremetal MACHINE=imx8mp-lpddr4-evk \
source real-time-edge-setup-env.sh \
-b build-imx-baremetal
```

Note: *Restarting a build environment*

Sometimes, a new terminal window is opened or the machine is rebooted after a build directory is set up. In such cases, use the setup environment script to set up the environment variables and run a build again. Sourcing the full `real-time-edge-setup-env.sh` is not required.

```
$ DISTRO=<distro> MACHINE=<machine> source setup-environment <build-dir>
```

6 Image deployment

Complete file system images are deployed to `<build directory>/tmp/deploy/images`. An image is, is, generally, specific to the machine set in the environment setup. Each image build creates a U-Boot, a kernel, and an image type based on the `IMAGE_FSTYPES` defined in the machine configuration file. Most machine configurations provide an SD card image (`.wic`) and a rootfs image (`.tar`). The SD card image contains a partitioned image (with U-Boot, kernel, rootfs, and other such files) suitable for booting the corresponding hardware.

6.1 Copy image to SD card

An SD card image file `.wic` contains a partitioned image (with U-Boot, kernel, rootfs, and other files) suitable for booting the corresponding hardware. To copy this image to an SD card, run the following commands:

```
$ zstd -d <image_name>.wic.zst
$ sudo dd if=<image name>.wic of=/dev/sd<disk> bs=1M conv=fsync
```

For more information about i.MX, see Section "Preparing an SD/MMC card to boot" in the i.MX Linux User's Guide https://www.nxp.com/docs/en/user-guide/UG10163.pdf.

For more information about Layerscape, see Layerscape Software Development Kit User Guide for Yocto.

7 Image deployment on eMMC

The default images of the Real-time Edge are for SD boot on most boards. But all the boards support multiple boot options. This chapter describes how to build and deploy images to the eMMC flash. Users can also use a similar way to deploy images to other boot media.

Supported platforms:

- `imx8mm-lpddr4-evk`
- `imx8mp-lpddr4-evk`
- `imx93evk`
- `imx93-9x9-lpddr4-qsb`
- `ls1028ardb`
- `ls1046ardb`

7.1 Boot options

7.1.1 LS1028ARDB boot options

LS1028ARDB supports the following boot options:

- FlexSPI NOR flash
- eMMC
- SD card (SDHC1)

The LS1028ARDB board supports user-selectable switches for evaluating different boot options for the LS1028A device. These are shown in the Table 2 ('0' is OFF, '1' is ON).

Table 2. LS1028ARDB boot options

Boot source	SW2[1:8]	SW3[1:8]	SW5[1:8]
FSPI NOR	1111_1000	1111_0000	0011_1001
SD Card (SDHC1)	1000_1000	1111_0000	0011_1001
eMMC	1001_1000	1111_0000	0011_1001

7.1.2 LS1046ARDB boot options

The LS1046ARDB board supports the following boot options:

- SD
- QSPI NOR flash

The RDB has user-selectable switches for evaluating different boot options for the LS1046A device. These options are listed in the Table 3.

Table 3. LS1046ARDB boot options ('0' is OFF, '1' is ON)

Boot source	SW3[1:8]	SW4[1:8]	SW5[1:8]
QSPI NOR flash0	01000110	00111011	00100010
QSPI NOR flash1	01001110	00111011	00100010
SD card	01000110	00111011	00100000

The LS1046A SDHC controller is connected to the on-board SDHC connector, SD card, and eMMC memories. The connector is muxed with eMMC or SD card as shown in Figure 1.

Note: *While using eMMC, the SD card must not be plugged in.*

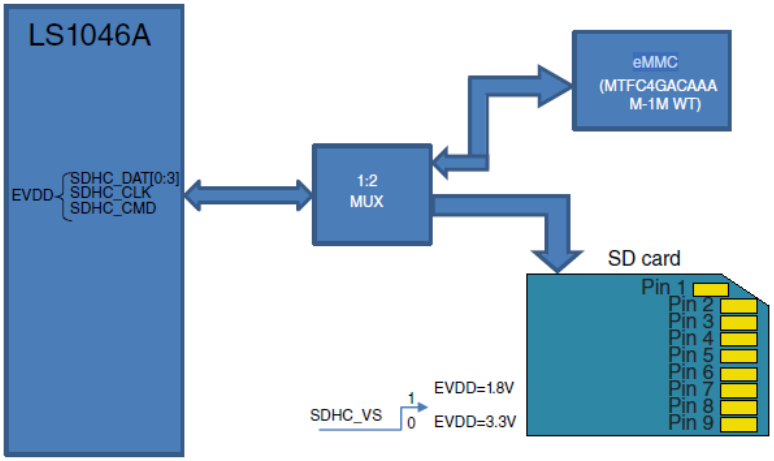

Figure 1. eSDHC connections

7.2 Building eMMC images for Layerscape platform

For Layerscape platform including `LS1028ARDB` and `LS1046ARDB`, we must build an eMMC image that contains eMMC bootloader and rootfs. The `nxp-real-time-edge-emmc` is the specific distro to build the eMMC wic image. i.MX related boards can use the `UUU` tool to burn eMMC bootloader and rootfs separately. There is no specific eMMC distro for i.MX boards.

First, use the below command to create the Yocto building environment.

```
$ cd yocto-real-time-edge
$ DISTRO=nxp-real-time-edge-emmc MACHINE=ls1028ardb source real-time-edge-setup-
env.sh -b build-qoriq-emmc
```

Then, enter the build directory and start to build the images.

For example, use the commands below to build ls1028ardb eMMC images:

```
$ source  setup-environment build-qoriq-emmc
$ DISTRO=nxp-real-time-edge-emmc MACHINE=ls1028ardb  bitbake nxp-image-real-
time-edge;
```

To build ls1046ardb eMMC images, use the steps below:

```
$ source  setup-environment build-qoriq-emmc
$ DISTRO=nxp-real-time-edge-emmc MACHINE=ls1046ardb  bitbake nxp-image-real-
time-edge;
```

After these steps are followed, all images are created and stored in `build-qoriq-emmc/tmp/deploy/images/BOARDS` folder. For example:

```
  build-qoriq-emmc/tmp/deploy/images/ls1028ardb/
       ├──atf
       │   ├──bl2_emmc.pbl
```

```
        │   ├──bl2_flexspi_nor.pbl
        │   ├──bl2_sd.pbl
        │   ├──fip_uboot.bin
        │   ├──srk.pri
        │   └──srk.pub
        ├──nxp-image-real-time-edge-ls1028ardb.rootfs.tar.zst
        ├──nxp-image-real-time-edge-ls1028ardb.rootfs.wic.zst
        ......
```

Code in the internal boot ROM loads the bl2 image from a boot device, such as NOR flash or SD/eMMC. Use either of the boot options listed below:

- `bl2_sd.pbl` for SD boot
- `bl2_emmc.pbl` for eMMC boot
- `bl2_flexspi_nor.pbl` for NOR flash boot

The bl2 image loads `fip_uboot.bin`, then enter U-Boot and boot the kernel.

`nxp-image-real-time-edge-ls1028ardb.rootfs.wic.zst` includes `bl2/uboot/kernel` and `rootfs`.

7.3 Deploying eMMC images with 'wic' image on Layerscape Platform

This section describes how to deploy eMMC image using '`wic`' images.

7.3.1 Burning 'wic' images

Real-time Edge `wic` images include `atf/uboot/Linux/rootfs` and their size is about 1.9 GB. This size is too large to download via U-Boot. Therefore, users must burn this image to eMMC flash under Linux.

To enter Linux:

- Use SD boot for LS1028ARDB. Use an SD card that was already flashed with the default Real-time Edge SD card image.
- Use QSPI boot for LS1046ARDB.

The following are the steps to burn `wic` image using LS1028ARDB as an example.

1. Use `zstd` command to decompress the wic image:

```
$ zstd -d nxp-image-real-time-edge-ls1028ardb.rootfs.wic.zst
```

2. In an SD card that has been flashed with the default Real-time Edge image, the default partition boot and rootfs are not large enough to store the `wic` images. Therefore, users must create a new, additional partition.
3. First, plug the SD card into a PC. Then, use the `parted` command or other tools to create a new partition. For example, use the command below to create a new partition:

```
$ parted /dev/sde
$ sudo parted /dev/sde
(parted) p
Model: Generic MassStorageClass (scsi)
Disk /dev/sde: 15.6GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:
Number  Start   End     Size    Type     File system  Flags
 1      16.8MB  191MB   174MB   primary  ext4         boot
 2      193MB   1937MB  1744MB  primary  ext4
```

```
(parted) mkpart primary 2G 16G
```

4. After exiting the Partition Manager, format the partition using the command:

```
 $ sudo mkfs.ext4 /dev/sde3
```

5. Copy the `wic` image to this partition using the command:

```
$ sudo mount  /dev/sde3 /mnt
$ sudo cp /media/data/tftp/ls1028ardb-emmc/nxp-image-real-time-edge-
ls1028ardb.rootfs.wic  /mnt
$ sync
$ sudo umount /mnt
```

6. Insert the SD card in the LS1028ARDB board and boot up the kernel.
7. In the kernel, one can see the SD card `mmcblk0` and eMMC card `mmcblk1`, and their partitions. See a sample log below.
   Note: *The below sample displays log for a case that used a previously installed eMMC. On a clean eMMC, there are no eMMC partitions mounted.*

```
root@ls1028ardb:~# lsblk
NAME         MAJ:MIN RM   SIZE RO TYPE MOUNTPOINTS
mtdblock0     31:0    0   256M  0 disk
mmcblk0      179:0    0  14.5G  0 disk
|-mmcblk0p1  179:1    0 166.4M  0 part /run/media/boot-mmcblk0p1
|-mmcblk0p2  179:2    0   1.6G  0 part /
`-mmcblk0p3  179:3    0  12.6G  0 part /run/media/mmcblk0p3
mmcblk1      179:32   0   7.1G  0 disk
|-mmcblk1p1  179:33   0   256M  0 part /run/media/mmcblk1p1
`-mmcblk1p2  179:34   0   500M  0 part
mmcblk1boot0 179:64   0     2M  1 disk
mmcblk1boot1 179:96   0     2M  1 disk
```

8. Unmount any mounted eMMC partitions as needed, and use the `dd` command to burn the eMMC device:

```
root@ls1028ardb:~# umount /dev/mmcblk1p1
root@ls1028ardb:~# umount /dev/mmcblk1p2
root@ls1028ardb:~# cd /run/media/mmcblk0p3
root@ls1028ardb:/run/media/mmcblk0p3# ls
lost+found       nxp-image-real-time-edge-ls1028ardb.rootfs.wic
root@ls1028ardb:/run/media/mmcblk0p3# dd if=nxp-image-real-time-edge-
ls1028ardb.wic  of=/dev/mmcblk1 bs=1M conv=fsync
1846+1 records in
1846+1 records out
1936594944 bytes (1.9 GB, 1.8 GiB) copied, 141.838 s, 13.7 MB/s
```

7.3.2 Boot up from eMMC

After the steps described in the previous section are followed, the `wic` the image is flashed into eMMC flash. This section describes the steps to boot the board via eMMC.

- Use enable eMMC boot under U-Boot using the commands below:
  - For LS1028ARDB:

```
=> qixis_reset emmc
```

  - For LS1046ARDB:

```
=> cpld reset sd
```

Or change the switch to enable eMMC boot.

The below code is an example of the eMMC boot log for LS1028ARDB:

Boot log from eMMC

```
U-Boot 2022.04+fsl+g3eb42755d5 (Feb 08 2023 - 16:44:45 +0000)
SoC:  LS1028AE Rev1.0 (0x870b0010)
Clock Configuration:
       CPU0(A72):1500 MHz  CPU1(A72):1500 MHz
       Bus:      400  MHz  DDR:      1600 MT/s
Reset Configuration Word (RCW):
       00000000: 3c004010 00000030 00000000 00000000
       00000010: 00000000 018f0000 0030c000 00000000
       00000020: 020031a0 00002580 00000000 00003296
       00000030: 00000000 00000008 00000000 00000000
       00000040: 00000000 00000000 00000000 00000000
       00000050: 00000000 00000000 00000000 00000000
       00000060: 00000000 00000000 200e705a 00000000
       00000070: bb580000 00000000
Model: LS1028A RDB Board
Board: LS1028AE Rev1.0-RDB, Version: A, boot from eMMC
FPGA: v6 (RDB)
SERDES1 Reference : Clock1 = 100.00MHz Clock2 = 100.00MHz
DRAM:  3.9 GiB
DDR    3.9 GiB (DDR4, 32-bit, CL=11, ECC on)
Using SERDES1 Protocol: 47960 (0xbb58)
PCIe1: pcie@3400000 Root Complex: no link
PCIe2: pcie@3500000 Root Complex: no link
Core:  42 devices, 22 uclasses, devicetree: separate
WDT:   Started watchdog@c000000 with servicing (60s timeout)
WDT:   Started watchdog@c010000 with servicing (60s timeout)
MMC:   FSL_SDHC: 0, FSL_SDHC: 1
Loading Environment from MMC... *** Warning - bad CRC, using default environment
EEPROM: Invalid ID (ff ff ff ff)
In:    serial
Out:   serial
Err:   serial
SEC0:  RNG instantiated
Net:
Warning: enetc-0 (eth0) using random MAC address - 5a:6a:5e:dc:66:34
eth0: enetc-0
Warning: enetc-2 (eth1) using random MAC address - 96:b5:ae:4a:1c:d1
, eth1: enetc-2, eth2: swp0, eth3: swp1, eth4: swp2, eth5: swp3
Hit any key to stop autoboot:  0
```

7.4 Deploying eMMC images with separate images on Layerscape Platform

This section describes how to burn separate images for eMMC. This section describes the procedure to burn the bl2 or fip image, create the boot/rootfs partition, and install rootfs manually.

The below steps use LS1046ARDB as an example.

7.4.1 Booting the board to U-Boot

For LS1028ARDB, first boot via SD.

List eMMC under U-Boot.

```
=> mmc list
FSL_SDHC: 0 (SD)
FSL_SDHC: 1
=> mmc dev 1
switch to partitions #0, OK
mmc1(part 0) is current device
=> mmc info
Device: FSL_SDHC
Manufacturer ID: 13
OEM: 4e
Name: Q2J55L
Bus Speed: 50000000
Mode: MMC High Speed (52MHz)
Rd Block Len: 512
MMC version 5.0
High Capacity: Yes
Capacity: 7.1 GiB
Bus Width: 8-bit
Erase Group Size: 512 KiB
HC WP Group Size: 8 MiB
User Capacity: 7.1 GiB WRREL
Boot Capacity: 2 MiB ENH
RPMB Capacity: 4 MiB ENH
Boot area 0 is not write protected
Boot area 1 is not write protected
```

For LS1046ARDB, use the QSPI U-Boot. If the QSPI flash is empty, re-burn the images.

For this, enter SD boot using the Real-time Edge distro release.

Then, re-burn the QSPI images using the below commands:

- Program QSPI NOR flash 1:

```
=> sf probe 0:0
```

- Flash `bl2_qspi.pbl`:

```
=> tftp 0xa0000000 bl2_qspi.pbl
=> sf erase 0x0 +$filesize && sf write 0xa0000000 0x0 $filesize
```

- Flash `fip_uboot.bin`:

```
=> tftp 0xa0000000 fip_uboot.bin
=> sf erase 0x100000 +$filesize && sf write 0xa0000000 0x100000 $filesize
```

- Change SW5 to *00100010* to select QSPI NOR flash0 boot and unplug the SD card.

Below is the displayed **U-Boot log** for LS1046ARDB:

```
U-Boot 2022.04+fsl+g3eb42755d5 (Feb 09 2023 - 02:27:05 +0000)

SoC:  LS1046AE Rev1.0 (0x87070010)
Clock Configuration:
       CPU0(A72):1600 MHz  CPU1(A72):1600 MHz  CPU2(A72):1600 MHz
       CPU3(A72):1600 MHz
       Bus:      600  MHz  DDR:      2100 MT/s  FMAN:     700  MHz
Reset Configuration Word (RCW):
       00000000: 0c150010 0e000000 00000000 00000000
       00000010: 11335559 40005012 40025000 c1000000
       00000020: 00000000 00000000 00000000 00238800
       00000030: 20124000 00003000 00000096 00000001
Model: LS1046A RDB Board
Board: LS1046ARDB, boot from QSPI vBank 4
CPLD:  V2.3
PCBA:  V2.0
SERDES Reference Clocks:
SD1_CLK1 = 156.25MHZ, SD1_CLK2 = 100.00MHZ
DRAM:  15.9 GiB (DDR4, 64-bit, CL=15, ECC on)
       DDR Chip-Select Interleaving Mode: CS0+CS1
Using SERDES1 Protocol: 4403 (0x1133)
Using SERDES2 Protocol: 21849 (0x5559)
PCIe1: pcie@3400000 Root Complex: no link
PCIe2: pcie@3500000 Root Complex: no link
PCIe3: pcie@3600000 Root Complex: no link
Core:  46 devices, 16 uclasses, devicetree: separate
NAND:  512 MiB
MMC:   FSL_SDHC: 0
Loading Environment from SPIFlash... SF: Detected s25fs512s with page size 256
 Bytes, erase size 256 KiB, total 64 MiB
OK
EEPROM: NXID v1
In:    serial
Out:   serial
Err:   serial
SEC0:  RNG instantiated
Net:   Fman1: Uploading microcode version 106.4.18
eth0: fm1-mac3, eth1: fm1-mac4, eth2: fm1-mac5, eth3: fm1-mac6, eth4: fm1-mac9,
 eth5: fm1-mac10
Hit any key to stop autoboot:  0
=> mmc info
Device: FSL_SDHC
Manufacturer ID: fe
OEM: 4e
Name: P1XXXX
Bus Speed: 50000000
Mode: MMC High Speed (52MHz)
Rd Block Len: 512
MMC version 4.5
High Capacity: Yes
Capacity: 3.6 GiB
Bus Width: 4-bit
Erase Group Size: 512 KiB
HC WP Group Size: 4 MiB
User Capacity: 3.6 GiB
```

```
Boot Capacity: 2 MiB ENH
RPMB Capacity: 128 KiB ENH
Boot area 0 is not write protected
```

7.4.2 Flashing Arm Trusted Firmware and U-Boot to eMMC

This section describes the steps for flashing Arm Trusted Firmware (ATF) and U-Boot to eMMC.

- First, select eMMC flash using the commands below:
  - For LS1046ARDB:

```
=> mmc dev 0
switch to partitions #0, OK
mmc0(part 0) is current device
```

  - For LS1028ARDB:

```
=> mmc dev 1
switch to partitions #0, OK
mmc1(part 0) is current device
```

- Then, burn the `bl2` image:

```
=> tftp 82000000 bl2_emmc.pbl
=> mmc write 82000000 8 <blk_cnt>
```

Where `<blk_cnt>` is the number of blocks in eMMC flash that must be written. It is calculated based on file size.
For example: if `bl2_sd.pbl` is loaded from the TFTP server and the number of bytes transferred is 53280 (d020 hex), then `<blk_cnt>` is calculated as below:
53280 /512 = 105 (69 hex)
For this example, use the command below:

```
=> mmc write 82000000 8 69
```

- Then, flash `fip_uboot.bin`:

```
=> tftp 82000000 fip_uboot.bin

=> mmc write 82000000 800 <blk_cnt>
```

- Use the commands below to boot to U-Boot:
  - For LS1046ARDB:

```
=> cpld reset sd
```

  - For LS1028ARDB:

```
=> qixis_reset emmc
```

Below is the **eMMC boot log** for LS1046ARDB:

```
U-Boot 2022.04+fsl+g3eb42755d5 (Feb 09 2023 - 02:27:05 +0000)
SoC:  LS1046AE Rev1.0 (0x87070010)
Clock Configuration:
       CPU0(A72):1800 MHz  CPU1(A72):1800 MHz  CPU2(A72):1800 MHz
       CPU3(A72):1800 MHz
       Bus:      600  MHz  DDR:      2100 MT/s  FMAN:     700  MHz
Reset Configuration Word (RCW):
       00000000: 0c150012 0e000000 00000000 00000000
       00000010: 11335559 40000012 60040000 c1000000
       00000020: 00000000 00000000 00000000 00238800
       00000030: 20124000 00003000 00000096 00000001
Model: LS1046A RDB Board
Board: LS1046ARDB, boot from SD
```

```
CPLD:  V2.3
PCBA:  V2.0
SERDES Reference Clocks:
SD1_CLK1 = 156.25MHZ, SD1_CLK2 = 100.00MHZ
DRAM:  15.9 GiB (DDR4, 64-bit, CL=15, ECC on)
       DDR Chip-Select Interleaving Mode: CS0+CS1
Using SERDES1 Protocol: 4403 (0x1133)
Using SERDES2 Protocol: 21849 (0x5559)
PCIe1: pcie@3400000 Root Complex: no link
PCIe2: pcie@3500000 Root Complex: no link
PCIe3: pcie@3600000 Root Complex: no link
Core:  46 devices, 16 uclasses, devicetree: separate
NAND:  512 MiB
MMC:   FSL_SDHC: 0
Loading Environment from MMC... OK
EEPROM: NXID v1
In:    serial
Out:   serial
Err:   serial

SEC0:  RNG instantiated
Net:
MMC read: dev # 0, block # 18432, count 128 ...
Fman1: Uploading microcode version 106.4.18
eth0: fm1-mac3, eth1: fm1-mac4, eth2: fm1-mac5, eth3: fm1-mac6, eth4: fm1-mac9,
 eth5: fm1-mac10
```

7.4.3 Installing rootfs

The following steps describe how to install the rootfs to the desired partition.

1. Prepare the rootfs. Insert a USB disk into a PC and copy the rootfs to an ext4 partition. For example:

```
$ zstd -d nxp-image-real-time-edge-ls1046ardb.rootfs.tar.zst
$ tar -xvf nxp-image-real-time-edge-ls1046ardb.rootfs.tar -C /mnt
```

2. Insert the USB disk to the board. Update `bootargs` parameter to set the correct rootfs device. For LS1046ARDB, use the command below:

```
=>setenv bootargs " root=/dev/sda3 rw rootwait console=ttyS0,115200
 earlycon=uart8250,mmio,0x21c0500"
```

3. Boot the kernel using the commands:

```
=> tftp 0x82000000 Image
=> tftp 0x8f000000 fsl-ls1046a-rdb-sdk.dtb
=> booti 0x82000000 - 0x8f000000
```

4. Create two partitions on eMMC flash using the commands below:

```
root@ls1046ardb:~# fdisk /dev/mmcblk0
Command (m for help): p
Disk /dev/mmcblk0: 3.6 GiB, 3867148288 bytes, 7553024 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xdb167106
Command (m for help): n
Partition type
   p   primary (0 primary, 0 extended, 4 free)
   e   extended (container for logical partitions)
Select (default p): p
Partition number (1-4, default 1):
First sector (2048-7553023, default 2048): 65536
Last sector, +/-sectors or +/-size{K,M,G,T,P} (65536-7553023, default
 7553023): +256M
Created a new partition 1 of type 'Linux' and of size 256 MiB.
Command (m for help): p
Disk /dev/mmcblk0: 3.6 GiB, 3867148288 bytes, 7553024 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xdb167106
Device         Boot Start    End Sectors  Size Id Type
/dev/mmcblk0p1      65536 589823  524288  256M 83 Linux
Command (m for help): n
Partition type
   p   primary (1 primary, 0 extended, 3 free)
   e   extended (container for logical partitions)
Select (default p):
Using default response p.
Partition number (2-4, default 2):
First sector (2048-7553023, default 2048): 589840
Last sector, +/-sectors or +/-size{K,M,G,T,P} (589840-7553023, default
 7553023):
Created a new partition 2 of type 'Linux' and of size 3.3 GiB.
```

```
Command (m for help): p
Disk /dev/mmcblk0: 3.6 GiB, 3867148288 bytes, 7553024 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xdb167106
Device         Boot  Start     End Sectors  Size Id Type
/dev/mmcblk0p1       65536  589823  524288  256M 83 Linux
/dev/mmcblk0p2      589840 7553023 6963184  3.3G 83 Linux
Command (m for help): w
The partition table has been altered.
Calling ioctl() to re-read partition table.
Syncing disks.
```

5. List the block device:

```
root@ls1046ardb:~# lsblk
NAME         MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
sda            8:0    1  7.5G  0 disk
|-sda1         8:1    1  2.6G  0 part
|-sda2         8:2    1  3.9M  0 part /run/media/sda2
`-sda3         8:3    1  4.9G  0 part /
mtdblock0     31:0    0  512M  0 disk
mtdblock1     31:1    0   64M  0 disk
mtdblock2     31:2    0   64M  0 disk
mmcblk0      179:0    0  3.6G  0 disk
|-mmcblk0p1  179:1    0  256M  0 part
`-mmcblk0p2  179:2    0  3.3G  0 part
mmcblk0boot0 179:32   0    2M  1 disk
mmcblk0boot1 179:64   0    2M  1 disk
```

6. Make the file system into the new partition.

```
$ mkfs.ext4 /dev/mmcblk0p1
$ mkfs.ext4 /dev/mmcblk0p2
```

7. Install Linux. Then, copy the kernel and `dtb` file from the USB disk to the first partition.

```
$  sudo mount /dev/mmcblk0p1  /mnt
$  cp Image fsl-ls1046a-rdb-sdk.dtb   /mnt/
$  cp ls1046ardb_boot.scr  /mnt/
```

8. Install rootfs from the USB disk to the second partition.

```
$ sudo mount /dev/mmcblk0p2  /mnt
$ tar -xvf nxp-image-real-time-edge-ls1046ardb.rootfs.tar  -C /mnt
```

7.4.4 Booting to kernel

Rebooting the board automatically enables it to boot to the kernel.

Users can also boot the kernel manually by using the command below:

```
=> setenv bootargs " root=/dev/mmcblk0p2 rw rootwait console=ttyS0,115200
 earlycon=uart8250,mmio,0x21c0500"
=> load mmc 0:1 0x82000000 Image
=> load mmc 0:1  0x8f000000 fsl-ls1046a-rdb-sdk.dtb
=> booti 0x82000000 - 0x8f000000
```

7.5 Deploying eMMC images on i.MX Platform

This section describes how to use `UUU` to deploy images to eMMC.

7.5.1 Using UUU

UUU (Universal Update Utility) is an evolution of MFGTools (also known as MFGTools v3). UUU is Freescale/NXP I.MX Chip image deploy tools. UUU have the same usage on both Windows and Linux. It means the same script works on both OSes.

1. **Downloading and installing UUU**
   - Download UUU version 1.5.125 or higher from https://github.com/NXPmicro/mfgtools/releases.
   - On windows, `uuu.exe` can be download and can be directly used.
   - On Ubuntu Linux, `uuu` can be download and stored to `/usr/local/bin` folder.
2. Connect a USB cable from a computer to the USB port on the board for the download link. (The USB port can be OTG/Type C or Micro-B, depending on the board).
3. Connect a USB cable from the OTG-to-UART port to the computer for console output.
4. Open a Terminal emulator program.
   The i.MX boards can communicate with a host server (Windows OS or Linux OS) using a serial cable. Common serial communication programs such as HyperTerminal, Tera Term, or PuTTY can be used. The below describes the serial terminal setup:
   a. 115200
   b. No parity
   c. 8 data bits
   d. 1 stop bit
5. Set the boot pin to Serial Download mode.
   There are various ways to enter the Serial Download mode. One way is to set the boot mode to boot from SD slot SD3 (set SW6 DIP switches 2 and 7 to on, and the rest are off). Do not insert the SD card into slot SD3, and power on the board. After the message "HID Compliant device" is displayed, the board enters Serial Download mode. Then insert the SD card into SD slot SD3. Another way to do this is to configure an invalid boot switch setting, such as setting all the DIP switches of SW6 to 'OFF'.
   The following table shows the boot switch settings for i.MX boards, which are used to enter serial download mode for the manufacturing tool. If the boot image in the boot media is not validated, the system also enters the Serial Download mode.

Table 4. Serial Downloader on i.MX 8M Mini EVK

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW1101	ON	OFF	OFF	OFF	X	X	X	X
SW1102	X	X	X	X	X	X	X	X

Table 5. Serial Downloader on i.MX 8M Plus EVK

Switch	D1	D2	D3	D4
SW4	OFF	OFF	OFF	ON

Table 6. Serial Downloader on i.MX 93 11x11 EVK

Switch	D1	D2	D3	D4
SW1301	ON	ON	OFF	OFF

Table 7. Serial Downloader on i.MX 93 9x9 QSB

Switch	D1	D2	D3	D4
SW601	OFF	OFF	OFF	ON

6. Burn the image
   To burn a single boot image and rootfs to eMMC, run the following command

```
uuu -b emmc_all <bootloader> <rootfs.wic.zst>
```

For example, use the commands below:

```
# for i.MX93 EVK
sudo uuu -b emmc_all imx-boot-imx93evk-sd.bin-flash_singleboot  nxp-image-
real-time-edge-imx93evk.rootfs.wic.zst
# for i.MX 8M Plus EVK
sudo uuu -b emmc_all imx-boot-imx8mpevk-sd.bin-flash_evk nxp-image-real-time-
edge-imx8mp-lpddr4-evk.rootfs.wic.zst
```

7. Boot the board via eMMC. For this, change the boot pin to eMMC boot mode and reset the board.
   The following table shows the boot switch settings to boot from eMMC boards.

Table 8. eMMC Boot mode on i.MX 8M Mini EVK

Switch	D1	D2	D3	D4	D5	D6	D7	D8
SW1101	OFF	ON	ON	ON	OFF	OFF	ON	OFF
SW1102	OFF	OFF	OFF	OFF	ON	OFF	ON	OFF

Table 9. eMMC Boot Mode on i.MX 8M Plus EVK

Switch	D1	D2	D3	D4
SW4	OFF	OFF	ON	OFF

Table 10. eMMC Boot Mode on i.MX 93 11x11 EVK

Switch	D1	D2	D3	D4
SW1301	OFF	OFF	OFF	OFF

Table 11. eMMC Boot mode on i.MX 93 9x9 QSB

Switch	D1	D2	D3	D4
SW601	OFF	OFF	ON	OFF

8 Building packages based on i.MX Yocto release

The following sections describe how to add packages of `meta-real-time-edge` into i.MX Yocto Project. The Table 12 describes the packages user should select in i.MX Yocto project.

Table 12. Selected packages on i.MX Yocto Project

Package	Recipe	Real-time Edge Linux	i.MX Linux
IGH EtherCAT MainDevice stack	igh-ethercat	Y	Y
LinuxPTP	linuxptp	Y	Y
OPC UA including OPC UA PubSub	libopen62541	Y	Y
real-time-edge-sysrepo	real-time-edge-sysrepo	Y	N
Jailhouse	jailhouse	Y	Y
Real-time Edge Baremetal	-	Y	N
Preempt-RT Linux	-	Y	N

8.1 Downloading i.MX Yocto release and Real-time Edge Yocto Layer

Install the i.MX Yocto project by referring to the User Guide:

1. Download i.MX Yocto release:

```
$ mkdir imx-yocto-bsp
$ cd imx-yocto-bsp
$ repo init -u https://github.com/nxp-imx/imx-manifest \
-b imx-linux-walnascar \
-m imx-6.12.20-2.0.0.xml
$ repo sync
```

2. Download Real-time Edge Yocto layer:

```
$ cd sources
$ git clone https://github.com/nxp-real-time-edge-sw/meta-real-time-edge.git
 \
-b Real-Time-Edge-v3.2-202507
```

3. Then, enable meta-real-time-edge layer in the i.MX image build.
   - Setup the build environment:

```
$ cd imx-yocto-bsp (The top directory of repo)
$ DISTRO=fsl-imx-wayland MACHINE=<machine name> source imx-setup-release.sh
 \ -b build-real-time-edge
```

   - Add `meta-real-time-edge` to `bblayers.conf` file under the specific build folder using the below commands:

```
$ cd build-real-time-edge (The build-directory)
$ vim conf/bblayers.conf
# Add the below setting; meta-cpan is required by one recipe of meta-real-
time-edge layer
BBLAYERS += "${BSPDIR}/sources/meta-real-time-edge"
```

8.2 Selecting packages of Real-time Edge Yocto layer

8.2.1 Packages from Real-time Edge Yocto layer

Some packages from the Real-time Edge Yocto layer must be added into the i.MX image separately. These are the following:

- `igh-ethercat`
- `real-time-edge-sysrepo`
- `libopen62541` (OPC UA including OPC UA PubSub)

To select the package, add it to the `IMAGE_INSTALL` in `local.conf` as below:

For example, use below commands to add the `igh-ethercat` package:

Adding `igh-ethercat` on i.MX 8M Mini EVK:

```
$ vim conf/local.conf
# Add package
IGH_ETHERCAT ??= " "
IGH_ETHERCAT:imx8mm-lpddr4-evk = " fec "
PACKAGECONFIG:append:pn-igh-ethercat = " ${IGH_ETHERCAT} "
IMAGE_INSTALL += " igh-ethercat "
```

Adding `igh-ethercat` on i.MX 8M Plus EVK:

```
$ vim conf/local.conf
# Add package
IGH_ETHERCAT ??= " "
IGH_ETHERCAT:imx8mp-lpddr4-evk = " fec "
PACKAGECONFIG:append:pn-igh-ethercat = " ${IGH_ETHERCAT} "
IMAGE_INSTALL += " igh-ethercat "
```

Note: *For i.MX Yocto release, the FEC Ethernet driver is built in the kernel and only the EtherCAT generic module can be used. To use a native EtherCAT-capable module on i.MX 8M Mini EVK or i.MX 8M Plus EVK, users must compile the FEC Ethernet driver as the kernel module by setting:*

- *"`CONFIG_FEC=m`" in kernel configuration*
- *`DEVICE_MODULES` to "`fec`" as described in Chapter 5.1.5, "IGH EtherCAT Setup" of Real-time Edge Software User Guide.*

Adding OPC UA (including OPC UA PubSub)

```
$ vim conf/local.conf
# Add package
# Select OPC UA example application
include ${BSPDIR}/sources/meta-real-time-edge/conf/distro/include/
libopen62541.inc
LIBOPEN62541_LOGLEVE = "300"
IMAGE_INSTALL += " libopen62541 "
```

8.2.2 Packages in i.MX Yocto layer

The packages that are in i.MX Yocto layer are overridden when adding `meta-real-time-edge` layer. If you require to keep the original package instead of using Real-time Edge packages, users must add these packages to "`BBMASK`" in the `bblayer.conf` as listed below.

- `avahi`
- `ethtool`
- `iproute2`
- `jailhouse`
- `linuxptp`
- `lldpd`
- `tsntool`

Below is the configuration that can be used to mask the packages in `bblayer.conf`:

```
$ vim conf/bblayers.conf
# Add the below setting
BBMASK += "meta-real-time-edge/recipes-extended/jailhouse/*.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/tsntool/tsntool_%.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/ethtool/ethtool_%.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/linuxptp/linuxptp_3.1.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/avahi/avahi_%.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/iproute2/iproute2_%.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/lldpd/lldpd_%.bbappend"
```

The above packages that can be selected running on i.MX Yocto layer.

For example:

```
$ vim conf/local.conf
# Add package
IMAGE_INSTALL +=  "  \
    linuxptp \
    jailhouse \
    iproute2 \
    lldpd \
    avahi-daemon \
    avahi-utils \
"
```

8.3 Building the image

After adding the package, users can start building images using the selected Real-time edge packages.

Build the i.MX image using the command below:

```
$ bitbake imx-image-multimedia
```

8.4 Running packages on i.MX release

1. **Running Jailhouse**
   Refer to Chapter 3.3.2, "Running PREEMPT_RT Linux in Inmate" and Chapter 3.3.3, "*Running Jailhouse Examples In Inmate*" of *Real-time Edge Software User Guide*.
2. **Running LinuxPTP**
   Refer to Chapter 4.3.5, "*Quick Start for IEEE 1588*" and Chapter 4.3.6, "*Quick Start for IEEE 802.1AS*" of *Real-time Edge Software User Guide*.
3. **Running IGH-EtherCAT**
   Refer to Chapter 5.1.5.2, "*IGH EtherCAT Setup*" of *Real-time Edge Software User Guide*.
4. **Running OPC UA including OPC UA PubSub**
   Refer to Chapter 5.3, "*OPC UA*" of *Real-time Edge Software User Guide*.

Refer https://www.nxp.com/design/software/development-software/real-time-edge-software:REALTIME-EDGE-SOFTWARE?tab=Documentation_Tab.

9 Building packages based on Layerscape Yocto release

This section describes how to add packages of meta-real-time-edge into the Layerscape Yocto Project. The Table 13 describes the packages user should select in Layerscape Yocto project.

Table 13. Selected packages on Layerscape Yocto project

Feature	Recipe	Real-time Edge Linux	Layerscape Linux
IGH EtherCAT MainDevice stack	igh-ethercat	Y	Y
LinuxPTP	linuxptp	Y	Y
OPC UA including OPC UA PubSub	libopen62541	Y	Y
real-time-edge-sysrepo	real-time-edge-sysrepo	Y	Y
Jailhouse	jailhouse	Y	Y
Real-time Edge Baremetal	-	Y	Y
Preempt-RT Linux	-	Y	Y

9.1 Downloading LSDK Yocto release and Real-time Edge Yocto layer

1. Download LSDK Yocto release using the commands below:

```
$ mkdir yocto-sdk
$ cd yocto-sdk
$ repo init -u https://github.com/nxp-qoriq/yocto-sdk \
-b walnascar -m default.xml
$ repo sync
```

2. Download Real-time Edge Yocto layer using the commands below (meta-cpan is required by one recipe):

```
$ cd sources
$ git clone https://github.com/nxp-real-time-edge-sw/meta-real-time-edge.git
 \
-b Real-Time-Edge-v3.2-202507
```

9.2 Enabling meta-real-time-edge layer in Layerscape Image Build

1. Setup the build environment:

```
$ . ./setup-env -m ls1028ardb
```

2. Add `meta-real-time-edge` to `bblayers.conf` under the specific build folder.

```
$ vim conf/bblayers.conf
# Add the below setting, meta-cpan is required by one recipe of layer meta-
real-time-edge
BBLAYERS += " ${TOPDIR}/../sources/meta-real-time-edge"
```

9.3 Selecting packages of Real-time Edge Yocto Layer

9.3.1 Packages from Real-time Edge Yocto layer

Some packages included in the Real-time Edge Yocto layer can be added into the Layerscape image separately. These packages are:

- igh_ethercat
- real-time-edge-sysrepo
- libopen62541 (OPC UA including OPC UA PubSub)

To select the package, add them to the IMAGE_INSTALL in local.conf as shown below:

Adding igh-ethercat:

```
$ vim conf/local.conf
# Add package
IGH_ETHERCAT ??= " "
IMAGE_INSTALLL:append = " igh-ethercat "
```

Adding real-time-edge-sysrepo

```
$ vim conf/local.conf
# Add package
REAL_TIME_EDGE_SYSREPO:ls1028ardb = ""
PACKAGECONFIG:append:pn-real-time-edge-sysrepo = "${REAL_TIME_EDGE_SYSREPO}"
IMAGE_INSTALL:append = " real-time-edge-sysrepo "
```

Adding OPC UA (including OPC UA PubSub)

```
$ vim conf/local.conf
# Add package
# Select OPC UA example application
include ../sources/meta-real-time-edge/conf/distro/include/libopen62541.inc
LIBOPEN62541_LOGLEVE = "300"
IMAGE_INSTALL:append = " libopen62541 "
```

9.3.2 Packages in Layerscape Yocto layer

The below packages that are in Layerscape Yocto layer are overridden when adding the meta-real-time-edge layer.

- avahi
- ethtool
- iproute2
- jailhouse
- linuxptp
- lldpd
- tsntool

If it is required to keep the original package instead of using Real-time Edge packages, add these packages to "BBMASK" in the bblayer.conf file as shown below.

```
$ vim conf/bblayers.conf
# Add the below setting
BBMASK += "meta-real-time-edge/recipes-extended/jailhouse/*.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/tsntool/tsntool_%.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/ethtool/ethtool_%.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/linuxptp/linuxptp_3.1.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/avahi/avahi_%.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/iproute2/iproute2_%.bbappend"
BBMASK += "meta-real-time-edge/recipes-extended/lldpd/lldpd_%.bbappend"
```

The above packages can be selected running on Layerscape Yocto layer. To select the package, the package name needs to added into "IMAGE_INSTALL".

For example:

```
$ vim conf/local.conf
# Add package
IMAGE_INSTALL:append = " \
    linuxptp \
    jailhouse \
    iproute2 \
    lldpd \
    avahi-daemon \
    avahi-utils \
"
```

9.4 Building the image

After adding the package, one can start to build an image with selected Real-time edge package.

Build Layerscape image using the command below:

```
$ bitbake fsl-image-networking
```

9.5 Running packages on Layerscape

1. **Running Jailhouse**
   The below process takes LS1028ARDB as an example.
   a. Get `fsl-ls1028a-rdb-jailhouse.dtb` from Real-time Edge Release. Other images are built according to above process.
   b. Run the below commands under U-Boot to boot up the board.

```
=> setenv bootargs "root=/dev/ram0 rw earlycon=uart8250,0x21c0500
 console=ttyS0,115200 ramdisk_size=0x10000000"
=> tftp 0x82000000 Image
=> tftp 0xa0000000 fsl-image-networking-ls1028ardb.ext2.gz.u-boot
=> tftp 0x90000000 fsl-ls1028a-jailhouse-rdb.dtb
=> booti 0x82000000 0xa0000000 0x90000000
```

   c. Transfer Linux kernel binary "Image" to folder `/usr/share/jailhouse/inmates/kernel/`.
   For other steps, refer to the following chapters in *Real-time Edge Software User Guide*:
   - Chapter 3.3.2, "*Running PREEMPT_RT Linux in Inmate*"
   - Chapter 3.3.3, "*Running Jailhouse Examples In Inmate*".
2. **Running LinuxPTP**
   Refer to the following chapters in *Real-time Edge Software User Guide* (refer Section 1.2):
   - *Chapter 4.3.5*, "*Quick Start for IEEE 1588*"
   - *Chapter 4.3.6*, "*Quick Start for IEEE 802.1AS*".
3. **Running IGH-EtherCAT**:
   Refer to *Chapter 5.1.3.2*, "*IGH EtherCAT Setup*" of *Real-time Edge Software User Guide*.
4. **Running OPC UA including OPC UA PubSub**:
   Refer to *Chapter 5.3*, "*OPC UA*" of *Real-time Edge Software User Guide*.

10 Improving compilation speed

This section describes how to improve the Yocto compilation speed.

10.1 Downloads

During a build, `bitbake` fetches source code from the Internet as described by the `SRC_URI` variable in the individual recipes that make up an image. Bitbake downloads this data to the path pointed to by the `DL_DIR` variable. By default, this path is the `downloads` directory inside the build directory `(${BSPDIR}/downloads/)`. Bitbake first looks in this directory to unpack the source next time.

If there are multiple builds on your machine, then by default, each has a '`downloads`' directory with similar (duplicated) contents. Likewise, if a user wants to test a clean build, then bitbake requires to re-download everything. In these scenarios, build times can be reduced by having a machine global `download` directory that all builds can make use of. This is achieved by setting the `DL_DIR` variable in your `local.conf` file to a path outside your build directory.

By default, the `DL_DIR` variable value located in the `local.conf` file is below:

```
# DL_DIR ?= "${BSPDIR}/downloads/"
```

It can be set to a machine's global `download` directory by changing the `DL_DIR` variable such as:

```
# DL_DIR ="/home/<user>/yocto/downloads"
```

During a build, `bitbake` also obtains sources from 'Source Control Managers' (revision control systems such as Git and SVN). By default, `bitbake` does not store this source in the `DL_DIR`. However, it is possible to configure Yocto to do so via the following addition to your `local.conf`:

```
# BB_GENERATE_MIRROR_TARBALLS ="1"
```

The command `BB_GENERATE_MIRROR_TARBALLS` instructs `bitbake` to create an archive for each source that it checks out and places that archive in the `downloads` directory. For subsequent builds, if the source is needed, it first looks in the `downloads` directory for an archive before attempting to fetch it via a Source Control Manager.

Making use of `DL_DIR` and `BB_GENERATE_MIRROR_TARBALLS` can improve the time it takes to build Yocto by reducing the amount of time taken to obtain sources from the internet. It also greatly improves the reliability and reproducibility of builds as the `SRC_URI` can help user to identify a URL that is temporarily unavailable (or hosted on a slow server).

10.2 Shared state cache

Yocto has a sophisticated mechanism for caching outputs of individual `bitbake` tasks and associating them with a hash that represents the inputs of the task. By keeping track of such data, Yocto can improve the speed of subsequent builds by reusing previously built outputs instead of building them again. Yocto stores this data in the path pointed to by the `SSTATE_DIR` variable. By default, this is the `sstate-cache` directory inside the build directory (`${BSPDIR}/sstate-cache`).

Much like the `DL_DIR`, the shared state cache can also be moved outside of the build directory to a machine global download directory that can be shared by all builds. This can significantly improve the speed of a build (and reduce disk space needed) by changing the configuration of `SSTATE_DIR`:

The original `SSTATE_DIR` variable value is stored in `local.conf`:

```
# SSTATE_DIR ?= "${BSPDIR}/sstate-cache"
```

It can be set to a machine global download directory by changing the `SSTATE_DIR` variable such as:

```
SSTATE_DIR ="/home/<user>/yocto/sstate-cache"
```

A clean build with a pre-existing sstate cache is likely to complete very rapidly as most of the tasks do not need to be executed. Instead, Yocto reuses the existing outputs from the sstate cache. This feature highlights the benefit of having a sstate cache shared for all builds on your local machine.

11 Building the real-time-edge image offline

Internet access is required in order to download all the source packages that are required to build a real-time-edge image. However, Internet access might not always be available while building with yocto. A developer may have a build machine with Internet restrictions. This section describes how to build real-time-edge image offline.

1. **Download source packages**
   At some point, users require a Yocto machine with Internet access to download all the sources that are required for a build. This step can be achieved using the '`runonly`' bitbake option. Users can download all the sources by using the following command:

   ```
   # bitbake nxp-image-real-time-edge --runonly=fetch
   ```

   Before issuing this command, relocate the sources by setting `DL_DIR`, and reuse the build artifacts by setting `SSTATE_DIR` in `local.conf`. For more information, refer Section 10.
2. **Disable Internet access**
   If you are working on a machine with no network access, you may tar the `downloads` directory (including `.done` files) from an Internet connected machine and copy the tarball to your build machine. Then, extract the source packages into the `downloads` directory. Internet access on the build machine can be disabled by setting the `BB_NO_NETWORK` flag in `local.conf`.

   ```
   # BB_NO_NETWORK = "1"
   ```
3. **Build the image**
   After that, build real-time-edge image by using the command:

   ```
   # bitbake nxp-image-real-time-edge
   ```

12 How to build without Yocto

This section describes how to download and build the real time edge image without using Yocto.

12.1 Building the kernel in a standalone environment

To build the kernel in standalone environment, perform the steps mentioned below:

1. Download the project source from the following path:
   https://github.com/nxp-real-time-edge-sw/real-time-edge-linux
2. Check it out to the tag:
   - `Real-Time-Edge-v3.2-202507`
3. Install the cross-toolchain on your host environment:
   For arm32 cross-toolchain:

```
wget https://developer.arm.com/-/media/Files/downloads/gnu/13.2.rel1/binrel/
arm-gnu-toolchain-13.2.rel1-x86_64-arm-none-linux-gnueabihf.tar.xz
tar xJf arm-gnu-toolchain-13.2.rel1-x86_64-arm-none-linux-gnueabihf.tar.xz -C
 $HOME
echo 'export PATH="$PATH:$HOME/arm-gnu-toolchain-13.2.Rel1-x86_64-arm-none-
linux-gnueabihf/bin"' >> ~/.bashrc
source ~/.bashrc
```

   For arm64 cross-toolchain:

```
wget https://developer.arm.com/-/media/Files/downloads/gnu/14.2.rel1/binrel/
arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-none-linux-gnu.tar.xz
tar xJf arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-none-linux-gnu.tar.xz -C
 $HOME
echo 'export PATH="$PATH:$HOME/arm-gnu-toolchain-14.2.rel1-x86_64-aarch64-
none-linux-gnu/bin"' >> ~/.bashrc
source ~/.bashrc
```

4. Configure the cross-toolchain on your host environment:
   Arm-v7A (32-bit) and Arm-v8A (64-bit) toolchain and environment are as below:
   - i.MX6

```
ARCH=arm
CROSS_COMPILE=arm-none-linux-gnueabihf-
```

   - i.MX8, i.MX 93, LS1028ARDB, LS1043ARDB, and LS1046ARDB

```
ARCH=arm64
CROSS_COMPILE=aarch64-none-linux-gnu-
```

5. Then, run the following commands to build kernel image:

```
/* build kernel image for i.MX 6ULL board */
 $ make imx_v7_defconfig
 $ make

/*
 * build kernel image for
 * i.MX 8M Mini LPDDR4 EVK Rev.C
 * i.MX 8M Plus LPDDR4 EVK
 * i.MX 8DXL LPDDR4 EVK
 * i.MX 93 EVK
 * i.MX 93 9x9 QSB
 * i.MX 93 14x14 EVK
 * i.MX 91 11x11 EVK
```

```
 * i.MX 95 15x15 EVK
 * i.MX 95 19x19 EVK
 */
 $ make imx_v8_defconfig
 $ make

/* build kernel image for
 * LS1028ARDB
 * LS1043ARDB
 * LS1046ARDB
 */
 $ make defconfig
 $ make lsdk.config
 $ make
```

6. Finally, the file `Image` is generated in the path `./arch/arm64/boot/`.

13 Note About the Source Code in the Document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Copyright 2021- 2025 NXP Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. Neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

14 Revision history

Table 14 summarizes the revisions to this document.

Document revision history

Document ID	Release date	Description
RTEDGEYOCTOUG v.3.2	29 July 2025	Updated for Real-time Edge Software Rev 3.2
RTEDGEYOCTOUG v.3.1	26 March 2025	Updated for Real-time Edge Software Rev 3.1
RTEDGEYOCTOUG v.3.0	17 December 2024	Updated for Real-time Edge Software Rev 3.0
RTEDGEYOCTOUG v.2.9	25 July 2024	• Updated for Real-time Edge Software Rev 2.9
RTEDGEYOCTOUG v.2.8	29 March 2024	• Updated for Real-time Edge Software Rev 2.8 • Added the section: Section 7.5
RTEDGEYOCTOUG v.2.7	18 December 2023	• Updated for Real-time Edge Software Rev 2.7
RTEDGEYOCTOUG v.2.6	28 July 2023	• Updated for Real-time Edge Software Rev 2.6 • Added imx93-9x9-lpddr4-qsb platform support
RTEDGEYOCTOUG v.2.5	30 March 2023	• Updated for Real-time Edge Software Rev 2.5 • Added Section 7
RTEDGEYOCTOUG v.2.4	16 December 2022	Updated for Real-time Edge Software Rev 2.4
RTEDGEYOCTOUG v.2.3	28 July 2022	Updated for Real-time Edge Software Rev 2.3
RTEDGEYOCTOUG v.2.2	29 March 2022	Updated for Real-time Edge Software Rev 2.2
RTEDGEYOCTOUG v.2.1	15 December 2021	Updated for Real-time Edge Software Rev 2.1
RTEDGEYOCTOUG v.2.0	30 July 2021	First release for Real-time Edge Software Rev 2.0

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Evaluation products — This evaluation product is intended solely for technically qualified professionals, specifically for use in research and development environments to facilitate evaluation purposes. It is not a finished product, nor is it intended to be a part of a finished product. Any software or software tools provided with an evaluation product are subject to the applicable license terms that accompany such software or software tools.

This evaluation product is provided on an "as is" and "with all faults" basis for evaluation purposes only and is not to be used for product qualification or production. If you choose to use these evaluation products, you do so at your risk and hereby agree to release, defend and indemnify NXP (and all of its affiliates) for any claims or damages resulting from your use. NXP, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this evaluation product remains with user.

In no event shall NXP, its affiliates or their suppliers be liable to user for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the evaluation product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that user might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP, its affiliates and their suppliers and user's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by user based on reasonable reliance up to the greater of the amount actually paid by user for the evaluation product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose and shall not apply in case of willful misconduct.

HTML publications — An HTML version, if available, of this document is provided as a courtesy. Definitive information is contained in the applicable document in PDF format. If there is a discrepancy between the HTML document and the PDF document, the PDF document has priority.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Amazon Web Services, AWS, the Powered by AWS logo, and FreeRTOS — are trademarks of Amazon.com, Inc. or its affiliates.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Layerscape — is a trademark of NXP B.V.

QorIQ — is a trademark of NXP B.V.

Contents