RTEDGEYOCTOUG

Real-time Edge Yocto Project User Guide

Rev. 2.4 — 16 December 2022 User guide

Document information

Information	Content
Keywords	RTEDGEYOCTOUG, Real-time Edge software Yocto layer, i.MX boards, Layerscape boards, Yocto project setup, image building, 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 focused 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 www.yoctoproject.org/docs/current/yocto-project-qs/yocto-project-qs.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 *IMX_YOCTO_PROJECT_USERS_GUIDE (IMXLXYOCTOUG)*.
- 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 come from 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

During the setup environment process of the Real-time Edge Yocto Project Community Board Support Package (BSP), the NXP End-User License Agreement (EULA) is displayed. To continue to use 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 packages.

1.2 Related documentation

- For more information about i.MX Yocto project, refer to IMX_YOCTO_PROJECT_USERS_GUIDE.pdf.
- For more information about LSDK Yocto project, refer to LSDKYOCTOUG.pdf.
- 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 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 5.15.52-rt47` 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 2022.04` 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-nodejs-lbt`
  - `real-time-edge-prl`
  - `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`: icc recipe resides in `recipes-extended` directory. It provides a tool to community between master/slave and slave/slave cores.
  - `jailhouse`: jailhouse recipe resides in `recipes-extended` directory. It is a partitioning hypervisor based on Linux.
- **Protocols recipes**
  - `canfestival`
  - `igh-ethercat`
  - `libnfc-nci`
  - `libopen62541`
  - `real-time-edge-libbee`

  - `real-time-edge-libblep`
- **Harpoon recipes**
  Harpoon recipes reside in 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 Yocto Project expected behavior in a Linux Host Machine, the packages and utilities described below must be installed. An important consideration is the hard disk space required in the host machine. For example, when 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

In order to build a Yocto Project, few packages should be installed; these are documented under the Yocto Project. Go to Yocto Project Quick Start and check for the packages that must be installed for your build machine.

Essential Yocto Project host packages are:

```
$ sudo apt-get install gawk wget git-core diffstat unzip
 texinfo gcc-multilib \
build-essential chrpath socat cpio python python3 python3-pip
 python3-pexpect \
xz-utils debianutils iputils-ping python3-git python3-jinja2
 libegl1-mesa libsdl1.2-dev \
pylint3 xterm rsync curl zstd lz4 libssl-dev
```

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 "`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-kirkstone \
-m real-time-edge-2.4.0.xml
$ repo sync
```

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 repo 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 below NXP hardware platforms.

- `imx6ull14x14evk`
- `imx8dxlb0-lpddr4-evk`
- `imx8mm-lpddr4-evk`
- `imx8mp-lpddr4-evk`
- `imx93evk`
- `ls1028ardb`
- `ls1012ardb`
- `ls1021atwr`
- `ls1021atsn`
- `ls1021aiot`
- `ls1043ardb`
- `ls1046ardb`
- `ls1046afrwy`
- `lx2160ardb-rev2`

Each build folder must be configured in such 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` – The normal image including Real-time and industrial package without BareMetal support.
- `nxp-real-time-edge-baremetal` – The BareMetal image (some boards do not support this distro).

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, which configures the build environment and it is stored in `meta-real-time-edge/conf/distro`.

`MACHINE`=<machine configuration name> is the machine name which 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 `local.conf` inside each build folder and the EULA acceptance query is no longer displayed for that build folder.

After the script runs, the working directory is the one just created by the script, specified with the `-b` option. A `conf` folder is created containing the files `bblayers.conf` and `local.conf`.

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 that can be used to build an image is `bitbake <image name>`. Additional parameters can be used for specific activities described below. Bitbake provides various useful options for developing a single component. User the command below to run with a `bitbake` parameter:

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

`<component>` **is a desired build package.**

The following table provides 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 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-kirkstone \
-m real-time-edge-2.4.0.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

This 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

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

1. 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, the setup environment script should be used to set up the environment variables and run a build again. The full `real-time-edge-setup-env.sh` is not needed.

```
$ source setup-environment <build-dir>
```

6 Image deployment

Complete filesystem images are deployed to `<build directory>/tmp/deploy/images`. An image is, for the most part, 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.

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

7 Building packages Based on i.MX Yocto release

This chapter describes how to add packages of meta-real-time-edge into i.MX Yocto Project.

Table 2. Selected packages on i.MX Yocto Project

Package	Recipe	Real-time Edge Linux	i.MX Linux
IGH EtherCAT master 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

Table 2. Selected packages on i.MX Yocto Project*...continued*

Package	Recipe	Real-time Edge Linux	i.MX Linux
Preempt-RT Linux	-	Y	N

7.1 Downloading i.MX Yocto Release and Real-time Edge Yocto Layer

Install i.MX Yocto project, 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-kirkstone \
-m imx-5.15.52-2.1.0.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-v2.4-202212
$ git clone https://github.com/rehsack/meta-cpan.git
```

Enabling meta-real-time-edge layer in i.MX Image Build

1. Setup 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
```

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

```
$ 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 += "${BSPDIR}/sources/meta-real-time-edge"
BBLAYERS += "${BSPDIR}/sources/meta-cpan"
```

7.2 Selecting Packages of Real-time Edge Yocto Layer

7.2.1 Packages from Real-time Edge Yocto layer

Some packages from Real-time Edge Yocto layer should 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 kernel and only the EtherCAT generic module can be used. In order to use native EtherCAT-capable module on i.MX 8M Mini EVK or i.MX 8M Plus EVK, user needs to compile FEC Ethernet driver as kernel module by setting "`CONFIG_FEC=m`" in kernel configuration and set `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 "
```

7.2.2 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 it is required to keep the original package instead of using Real-time Edge packages, you 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 \
"
```

7.3 Building the image

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

Build the i.MX image using the command below:

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

7.4 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` 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.

8 Building packages based on Layerscape Yocto release

This chapter describes how to add packages of meta-real-time-edge into the Layerscape Yocto Project.

Table 3. Selected packages on Layerscape Yocto project

Feature	Recipe	Real-time Edge Linux	Layerscape Linux
IGH EtherCAT master stac	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

8.1 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 kirkstone -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-v2.4-202212
$ git clone https://github.com/rehsack/meta-cpan.git
```

8.2 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"
BBLAYERS += " ${TOPDIR}/../sources/meta-cpan"
```

8.3 Selecting packages of Real-time Edge Yocto Layer

8.3.1 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 = ""
REAL_TIME_EDGE_SYSREPO:ls1021atsn = "real-time-edge-sysrepo-tc"
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 "
```

8.3.2 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 you require 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 \
"
```

8.4 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
```

8.5 Running packages on Layerscape

1. **Running Jailhouse**
   The below process takes LS1028ARDB as 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*:
- *Chapter 4.1.5*, " *Quick Start for IEEE 1588*"
- *Chapter 4.1.6*, "*Quick Start for IEEE 802.1AS*".

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*.

9 Revision history

The table below describes the revisions to this document.

Document revision history

Date	Revision	Cross-reference	Changes
16 December 2022	2.4	-	Updated for Real-time Edge Software Rev 2.4
28 July 2022	2.3	-	Updated for Real-time Edge Software Rev 2.3
29 March 2022	2.2	-	Updated for Real-time Edge Software Rev 2.2
15 December 2021	2.1	-	Updated for Real-time Edge Software Rev 2.1
30 July 2021	2.0	-	First release for Real-time Edge Software Rev 2.0

10 Legal information

10.1 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.

10.2 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 http://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 data sheet 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.

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.

10.3 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.

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 or registered trademarks of Arm Limited (or its subsidiaries) 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.

i.MX — is a trademark of NXP B.V.

Contents

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

Date of release: 16 December 2022
Document identifier: RTEDGEYOCTOUG