UG10199

NXP MPU Cortex-A Core Zephyr User Guide

Rev. 1.0 — 26 March 2025 **User guide**

Document information

Information	Content
Keywords	UG10199, NXP MPUs, Cortex-A core, Zephyr User Guide, i.MX 8M Mini EVK, i.MX 8M Plus EVK, i.MX 8M Nano EVK, i.MX 93 EVK, and i.MX 95 EVK boards, Zephyr driver information, driver testing and examples
Abstract	This document explains the hardware setup, use cases, and commands to run Zephyr on the supported NXP MPU Cortex A hardware platforms.

1 Resources

This section describes the Github repositories and official documents to support Zephyr development.

1.1 Zephyr Github repositories

Table 1. Github Repo for Zephyr

Name	Repo	Branch
Zephyr	https://github.com/zephyrproject-rtos/zephyr	main
hal_nxp	https://github.com/zephyrproject-rtos/hal_nxp	master

1.2 Zephyr official documents

The official Zephyr documents are available on the URL: https://docs.zephyrproject.org/latest/

See the Introduction to Zephyr for a high-level overview. To start developing Zephyr applications, refer to the Zephyr Getting Started Guide. Refer to Release-Management for official Zephyr release milestone dates.

2 Zephyr development

Refer to Getting Started Guide to:

- Set up a command-line Zephyr development environment on Ubuntu, Mac OS, or Windows (instructions for other Linux distributions are discussed in Install Linux Host Dependencies).
- Get the source code.
- Build, flash, and run a sample application.

3 Zephyr kernel

3.1 Hardware Model V2

After Zephyr v3.6 release, Zephyr kernel transitions to Hardware Model V2. This new model overhauls the way both SoCs and boards are named and defined and adds support for the important features. These are the main features:

- Support for multi-core, multi-arch AMP (Asymmetrical Multi Processing) SoCs
- Support for multi-SoC boards
- Support for reusing the SoC and board Kconfig trees outside of the Zephyr build system
- Support for advanced use cases with Sysbuild (System build)
- Removal of all existing arbitrary and inconsistent uses of Kconfig and folder names

Zephyr's hardware support hierarchy has the following levels, from most to least specific:

- board, which has one or more
- SoC, each of which optionally belong to a
- SoC series, which in turn may optionally belong to an
- SoC family. Each SoC has one or more
- CPU cluster, each containing one or more
- CPU core, of a particular
- architecture

You can visualize the hierarchy in the figure below:

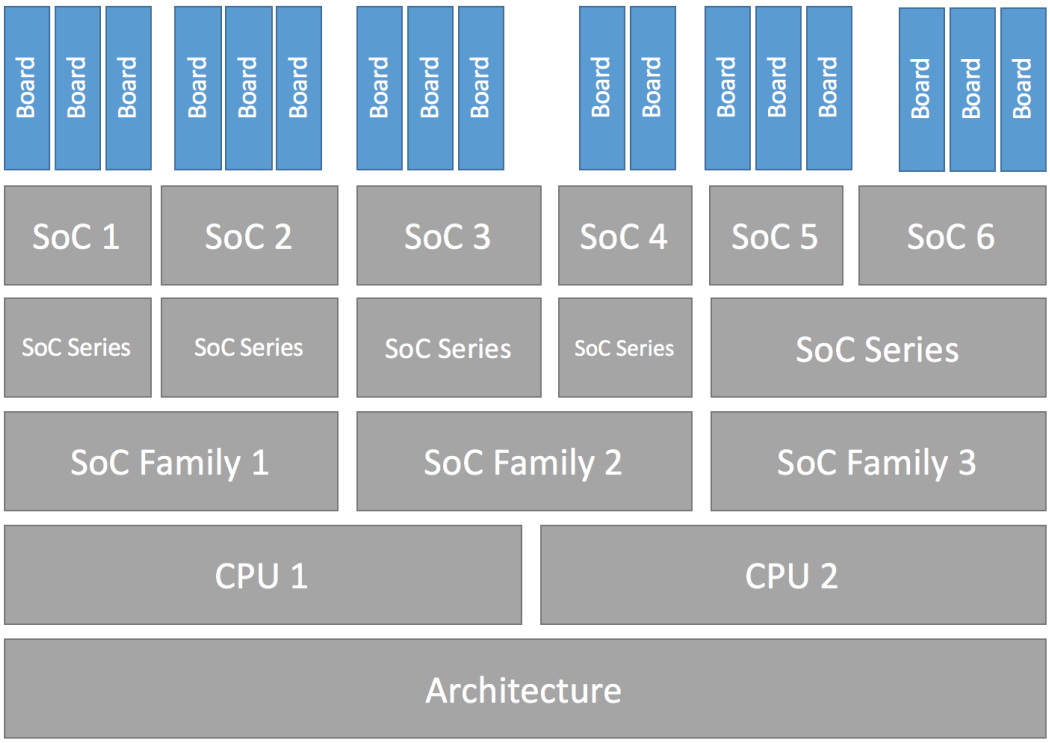

Figure 1. Hardware support hierarchy

Refer to Zephyr documentation for more details.

Important:

- *In this document, "**HWMV1**" refers to Hardware Model V1 supported in Zephyr release v3.6 and previous releases.*
- *"**HWMV2**" refers to Hardware Model V2 supported after Zephyr release v3.6.*

.

4 NXP MPUs supported in Zephyr

4.1 i.MX 93

4.1.1 Overview

NXP provides the i.MX 93 family of processors that are power-optimized for smart home, building control, contactless HMI, IoT edge, automotive, and industrial applications.

The i.MX 93 processor includes the powerful dual Arm™ Cortex-A55 processors with speeds up to 1.7 GHz integrated with a NPU that accelerates the machine learning inference. A general-purpose Arm™ Cortex™-M33 core running up to 250 MHz provides for real-time and low-power processing. Robust control networks are possible via a CAN-FD interface. Also, dual 1 Gbit/s Ethernet controllers, one supporting Time Sensitive Networking (TSN), drive gateway applications with low latency.

The i.MX 93 automotive qualified part is useful for applications such as:

- Driver monitoring system (DMS)
- Cost optimized gateway
- General purpose compute

Refer to NXP official website for more details: i.MX 93 Applications Processors Family | NXP Semiconductors

4.1.2 Zephyr SoCs and board

From Zephyr 3.7 release, hardware model v1 (HWMV1) is updated to hardware model v2 (HWMV2).

Table 2. HWMV2

Parameter	Directory/File Name
SoC source code directory	`soc/nxp/imx/imx9/`
Board source code directory	`boards/nxp/imx93_evk/`
Supported board name and variants	`imx93_evk/mimx9352/a55`

Table 3. HWMV1

Parameter	Directory/File Name
SoC source code directory	`soc/arm64/nxp_imx/mimx9/`
Board source code directory	`boards/arm64/mimx93_evk/`
Supported board name and variants	`mimx93_evk_a55`

4.1.3 Peripheral drivers

Table 4. Peripheral drivers on i.MX 93

Peripheral	Driver	Reference
GIC	GIC v3	5.3 Interrupt-Controller - GIC
Clock	CCM	5.7 Clock – CCM Rev2
IOMUX Controller	Pinctrl IMX	5.4 IOMUX – pinctrl_imx
UART	LPUART	5.2 UART - LPUART

Table 4. Peripheral drivers on i.MX 93*...continued*

Peripheral	Driver	Reference
GPIO	rgpio	5.11 GPIO - RGPIO
I2C	LPI2C	5.14 I2C - LPI2C
TPM Counter	TPM	5.10 Counter - TPM
Ethernet - ENET	ENET	5.15 Ethernet - ENET

4.1.4 Building Zephyr

Take `samples/hello_world` application as example:

HWMV2:

```
west build -p always -b imx93_evk/mimx9352/a55 samples/hello_world/
```

HWMV1:

```
west build -p always -b mimx93_evk_a55 samples/hello_world/
```

Then Zephyr binary image "`zephyr.bin`" and elf image "`zephyr.elf`" can be located in the `zephyr/build/zephyr/` directory.

4.1.5 Running Zephyr on the i.MX 93 board

Zephyr on Cortex-A Core can be booted up or stopped on a specified Cortex-A Core by using the U-Boot commands or remoteproc under Linux.

4.1.5.1 Board setup and software preparation (i.MX 93 EVK)

To run Zephyr, prepare the i.MX 93 EVK board using the steps below:

1. **UART Console Setup**
   Connect USB debug connector J1401 to the PC. The PC enumerates four COM ports when the USB cable is plugged into J1401. For example, on a Linux host, the four COM ports could be `/dev/ttyUSB0 ~ /dev/ttyUSB3`. If multiple USB serial cables are plugged into the same PC, the ID of tty USB device might be different. Therefore, users must check the kernel log after i.MX 93 EVK debug cable is plugged into the PC. For example, the following log shows that the COM ports should be `/dev/ttyUSB0 ~ /dev/ttyUSB3`.

```
linux:~$ sudo dmesg | tail
[272709.527874] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB0
[272709.528196] ftdi_sio 3-2.1.2:1.1: FTDI USB Serial Device converter
 detected
[272709.528266] usb 3-2.1.2: Detected FT4232H
[272709.528582] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB1
[272709.528888] ftdi_sio 3-2.1.2:1.2: FTDI USB Serial Device converter
 detected
[272709.528957] usb 3-2.1.2: Detected FT4232H
[272709.529217] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB2
```

```
[272709.529451] ftdi_sio 3-2.1.2:1.3: FTDI USB Serial Device converter
 detected
[272709.529514] usb 3-2.1.2: Detected FT4232H
[272709.529780] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB3
```

The third port (it is `/dev/ttyUSB2` in the above example) is used for U-Boot and Linux, and the fourth port (it is `/dev/ttyUSB3` in the above example) is used for Zephyr debug console by default. Developers can use minicom, Putty, Tera Term, Xshell, or other terminal tools to open these two UART consoles. The settings used are the **115200** Baud rate, **8 bit** data bits, **1** stop bits, and **none** parity.

2. **Deploying the pre-built SD card image**
   Zephyr on Cortex-A Core can be booted up or stopped on specified Cortex-A Core by using the U-Boot commands or remoteproc under Linux. U-Boot commands and remoteproc to start or stop Cortex-A Core Zephyr are supported in Real-time Edge software release v3.0 and late releases (https://www.nxp.com/rtedge), so a quick method to setup software environment on i.MX 93 EVK is to deploy the pre-build image (.wic image) to SD card and boot from it directly.
3. Follow the following steps to prepare an SD/MMC card to boot up an i.MX board using a Linux host machine.
4. Download latest Real-time Edge release pre-build image for the EVK board from https://www.nxp.com/rtedge

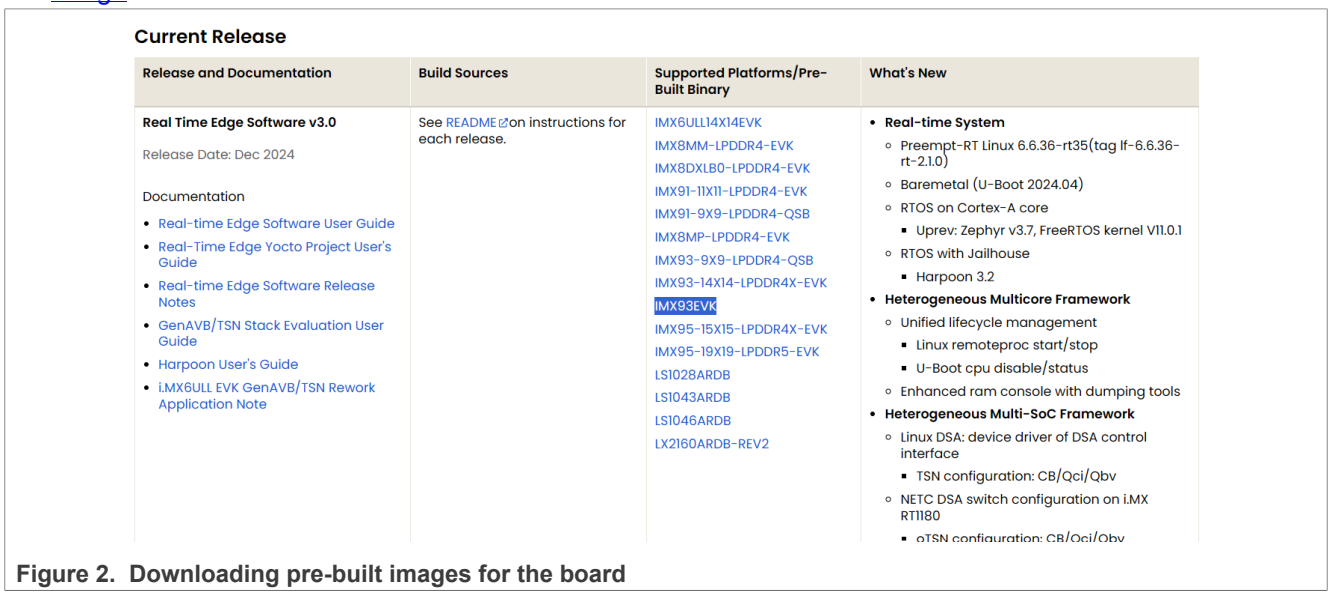

Figure 2. Downloading pre-built images for the board

5. Then a zip file “Real-time_Edge_vx.x_IMX93EVK.zip” (x.x is replaced with release version) is downloaded, extract the `.wic` file named “`nxp-image-real-time-edge-imx93evk.rootfs.wic`” from the path “`Real-time_Edge_vx.x_IMX93EVK/real-time-edge/nxp-image-real-time-edge-imx93evk.rootfs.wic.zst`” in the zip file.
   The SD card image (with the extension .wic) contains U-Boot, the Linux image and device trees, and the rootfs for a 4 GB SD card. The image can be installed on the SD card with one command if flexibility is not required. Carry out the following command to copy the SD card image to the SD/MMC card. Change sdx below to match the one used by the SD card.

```
$ sudo dd if=<image name>.wic of=/dev/sdx bs=1M && sync
```

The entire contents of the SD card are replaced. If the SD card is larger than 4 GB, the additional space is not accessible.

6. Then, insert the SD card back to the EVK board. Configure the EVK board to boot from SD card by switching SW1301[3:0] to be “0010”.

Power up the board and locate the TF-A and U-Boot log from the third port (it is `/dev/ttyUSB2` in the above example).

4.1.5.2 Booting Zephyr by using U-Boot commands

Multiple U-Boot commands can be used to start or stop Zephyr on the Cortex-A core.

4.1.5.2.1 "cpu" command

Following is the help information of the U-Boot `cpu` command:

```
u-boot=> cpu
```

```
cpu - Multiprocessor CPU boot manipulation and release
```

Usage:

- `cpu <num> reset` - Resets cpu <num>
- `cpu status` - Displays the status of all cpus
- `cpu <num> status` - Displays the status of cpu <num>
- `cpu <num> disable` - Disables the status of cpu <num>
- `cpu <num> release <addr> [args]` - Releases cpu <num> at <addr> with [args]

Except the "`cpu <num> reset`" command, all other cpu commands have been implemented on i.MX 8M Mini.

In general, U-Boot runs on the master Core (Core0) of Cortex-A cores. Therefore you can use "`cpu <num> release <addr> [args]`" to start RTOS on any other slave Cortex-A core (the Cores except Core0) from the specified memory address. For this purpose, you must load the RTOS binary images into the corresponding memory space, and use the "`cpu <num> disable`" command to power off any slave core that runs RTOS. Use the commands "`cpu status`" or "cpu <num>" status to check the status (running or power off) for all or the specified Cortex-A cores.

4.1.5.2.2 "go" command

The following is the help information of the U-Boot `go` command:

```
U-Boot=> go
```

go - starts the application at address 'addr'

Usage:

```
go addr [arg ...]
```

- starts the application at address '`addr`'

Passing '`arg`' as arguments

The "`go`" command can be used to start the application or RTOS running on the master Core (Core0). The impact is not returned back to the U-boot command line.

In summary, you can use U-Boot command "`go`" to boot the RTOS from Core0 or use U-Boot command "`cpu`" to boot or power off the RTOS running on the other Cortex-A cores except Core0.

4.1.5.3 Booting Zephyr by using Linux RemoteProc

Remoteproc (Remote Processor Framework) under Linux is another Cortex-A core Zephyr RTOS management tool in addition to the U-Boot commands in U-Boot. Using remoteproc can dynamically start or stop Zephyr under Linux. The limitation of this Cortex-A core Zephyr life cycle management method is that Linux must be running on at least single Cortex-A Core in order to manage Zephyr running on any of the other Cortex-A cores.

Remoteproc is used to control the remote processors that might run different instances of operation system. For example, it can be used for power on, loading firmware, or switching off the power of the remote processor. For SMP Linux which it runs on multiple Cortex-A Cores, each Cortex-A Core can be used as a remote processor, and can use remoteproc to bring up or bring down another RTOS instance on it. In order to run another RTOS on it, remoteproc first removes this Cortex-A Core from SMP Linux by using CPU hotplug. Then, brings it to be down. It then brings it up again with a new RTOS instance to run it. Remoteproc can also bring this CPU core running RTOS to be down and then plug it back to SMP Linux by using CPU hotplug.

- **Linux Device Tree Configuration**:
  In order to use remoteproc to manage life cycle on Cortex-A Core, you must add device nodes in Linux dts. For example, the following dts nodes are defined in `imx93-rproc-ca55.dtsi` (provided in Real-time Edge Linux kernel: https://github.com/nxp-real-time-edge-sw/real-time-edge-linux) . This file defines single remoteproc instances, it use fsl, cpus-bits defines CPU core bitmask to specify which CPU core(s) can be managed by this instance. The memory-region specifies the reserved memory space used by RTOS to be run on this instance.

```
ca55_1: remoteproc-ca55-1 {
  compatible = "fsl,imx-rproc-psci"; /* bitmask:0b10, assign A55 Core 1 */
  fsl,cpus-bits = <0x2>;
  memory-region = <&rtos_ca55_reserved>;
};
```

- **Hardware Resource Allocation between RTOS and Linux**
  In order to run a flexible AMP system with Multiple RTOS and Linux running together on the single SoC simultaneously, the hardware resources need to be allocated to different OS carefully to avoid conflicts, the hardware resources include memory and peripherals. Refer to `imx93-11x11-evk-multicore-rtos.dts` in Real-time Edge Linux (https://github.com/nxp-real-time-edge-sw/real-time-edge-linux) as an example, the memory used by RTOS need to be added to reserved-memory in Linux dts, and Linux dts nodes UART4 should be disabled as it is used as RTOS debug console.

- **GIC Configuration to run multiple RTOS on Cortex-A Cores**
  When running multiple OSes on different Cortex-A Cores on one platform, you must avoid GIC to be re-configured to crash the OS that has already been started. For Zephyr, you need to enable "`CONFIG_GIC_SAFE_CONFIG=y`". For the Linux kernel, you must enable "`CONFIG_GIC_GENTLE_CONFIG=y`".

- **Using remoteproc to start or stop RTOS on Cortex-A Core**

1. **Start Linux** In order to run RTOS, must use a device tree which is enabled remoteproc and has compatible resource allocation between RTOS and Linux. In Real-time Edge Linux, "`imx93-11x11-evk-multicore-rtos.dtb`" can be used to run the Zephyr `hello_world` application.
   To do this, when U-Boot is executing, stop at the U-Boot prompt with a terminal emulator connected to the serial port and execute the following commands:

```
u-boot=> setenv fdtfile imx93-11x11-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run bsp_bootcmd
```

2. **Using RemoteProc to start or stop Zephyr** When Linux boots up, log in Linux. Then use "`scp`" or other methods to download or copy `zephyr.elf` to the directory: `/lib/firmware/`
   The remoteproc sys portal is available at the following directory:

`/sys/devices/platform/remoteproc-ca55-1/remoteproc/remoteproc0`

3. Then, use the following command to start Zephyr:

```
root@imx93evk:~# echo zephyr.elf > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/firmware
root@imx93evk:~# echo start > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

Zephyr log will display on the Zephyr UART console. If needed, use the following command to stop RTOS running on Core1:

```
root@imx93evk:~# echo stop > /sys/devices/platform/remoteproc-ca55-1/
remoteproc/remoteproc0/state
```

- **Cortex-A Core dynamic allocation between Linux and the RTOS**
  Before running RTOS by using remoteproc, all Cortex-A Cores are used by the SMP Linux kernel by default. After running RTOS on the specified Cortex-A Core by using remoteproc, these Cortex-A core(s) are hot removed from the SMP Linux kernel and RTOS runs on them. After using the remoteproc `stop` command to stop the RTOS running on Cortex-A Core, these CPU cores once used by RTOS are hot added back to the SMP Linux kernel. Use the following command to check which CPU Core is used by the SMP Linux kernel currently:

```
root@imx93evk:~# cat /proc/cpuinfo
```

4.2 i.MX 8M Plus

4.2.1 Overview

The i.MX 8M Plus family focuses on machine learning and vision, advanced multimedia, and industrial automation with high reliability. It is built to meet the needs of Smart Home, Building, City and Industry 4.0 applications.

- Powerful quad or dual Arm Cortex-A53 processor with a Neural Processing Unit (NPU) operating at up to 2.3 TOPS.
- Dual image signal processors (ISP) and two camera inputs for an effective advanced vision system.
- The multimedia capabilities include video encode (including h.265) and decode, 3D/2D graphic acceleration, and multiple audio and voice functionalities.
- Real-time control with Cortex-M7. Robust control networks supported by dual CAN FD and dual Gigabit Ethernet with Time Sensitive Networking (TSN).
- High industrial reliability with DRAM inline ECC.

Refer to NXP Official website for more details: i.MX 8M Plus | Cortex-A53/M7 | NXP Semiconductors

4.2.2 Zephyr SoC and Board

From Zephyr 3.7 release, Zephyr SoC and Board have changed from hardware model v1 (HWMV1) to hardware model v2(HWMV2).

Table 5. HWMV2:

Parameter	Path
SoC source code directory	soc/nxp/imx/imx8m/
Board source code directory	boards/nxp/imx8mp_evk/
Supported board name and variants	imx8mp_evk/mimx8ml8/a53 imx8mp_evk/mimx8ml8/a53/smp

Table 6. HWMV1

Parameter	Parameter
SoC source code directory	soc/arm64/nxp_imx/mimx8m/
Board source code directory	boards/arm64/mimx8mp_evk/
Supported board name and variants	mimx8mp_evk_a53mimx8mp_evk_a53_smp

4.2.3 Peripheral drivers

Table 7. Peripheral drivers on i.MX 8M Plus

Peripheral	Driver	Reference
GIC	GIC v3	5.3 Interrupt-Controller - GIC
Clock	CCM	5.6 Clock – CCM
IOMUX Controller	Pinctrl IMX	5.4 IOMUX – pinctrl_imx
UART	IUART	5.1 UART - IUART
GPIO	IGPIO	5.12 GPIO - IGPIO

Table 7. Peripheral drivers on i.MX 8M Plus*...continued*

Peripheral	Driver	Reference
I2C	II2C	5.13 I2C - II2C
TPM Counter	GPT	5.9 Counter - GPT
Ethernet - ENET	ENET	5.15 Ethernet - ENET

4.2.4 Building Zephyr application

Take the `samples/hello_world` application as an example:

HWMV2:

```
west build -p always -b imx8mp_evk/mimx8ml8/a53 samples/hello_world/
```

HWMV1:

```
west build -p always -b mimx8mp_evk_a53 samples/hello_world/
```

Then Zephyr binary image "`zephyr.bin`" and elf image "`zephyr.elf`" can be located in the `zephyr/build/zephyr/` directory.

4.2.5 Running Zephyr application on the i.MX 8M Plus board

Zephyr on Cortex-A Core can be booted up or stopped on specified Cortex-A Core by using the U-Boot commands or remoteproc under Linux.

4.2.5.1 Board setup and software preparation (i.MX 8M Plus)

Prepare the i.MX 8M Plus EVK board to run Zephyr:

1. **UART Console Setup** Connect USB debug connector J23 to PC, PC enumerates four COM ports when the USB cable is plugged into J23. For example on Linux host, the four COM ports could be `/dev/ttyUSB0 ~ /dev/ttyUSB3`, the ID of a `ttyUSB` device could be different if multiple USB serial cables are plugged into the same PC. Therefore, you can check the kernel log after the i.MX EVK debug cable is plugged into the PC. For example, the following log shows the COM ports should be `/dev/ttyUSB0` ~ `/dev/ttyUSB3`.

```
linux:~$ sudo dmesg | tail
[272709.527874] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB0
[272709.528196] ftdi_sio 3-2.1.2:1.1: FTDI USB Serial Device converter
 detected
[272709.528266] usb 3-2.1.2: Detected FT4232H
[272709.528582] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB1
[272709.528888] ftdi_sio 3-2.1.2:1.2: FTDI USB Serial Device converter
 detected
[272709.528957] usb 3-2.1.2: Detected FT4232H
[272709.529217] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB2
[272709.529451] ftdi_sio 3-2.1.2:1.3: FTDI USB Serial Device converter
 detected
[272709.529514] usb 3-2.1.2: Detected FT4232H
[272709.529780] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB3
```

The third port (it is `/dev/ttyUSB2` in the above example) is used for U-Boot and Linux. The fourth port (it is /dev/ttyUSB3 in the above example) is used for Zephyr debug console by default. Developers can use minicom, Putty, Tera Term, Xshell, or other terminal tools to open these two UART consoles with the 115200 Baud rate, 8-bit data bits, 1 stop bit, and none parity.

2. **Deploying the pre-built SD card image** Zephyr on Cortex-A Core can be booted up or stopped on specified Cortex-A Core by using the U-Boot commands or remoteproc under Linux. U-Boot commands and remoteproc to start or stop Cortex-A Core Zephyr are supported in Real-time Edge software release in v3.0 and late releases (https://www.nxp.com/rtedge), so a quick method to setup software environment on i.MX 8M Plus EVK is to deploy the pre-build image (.wic image) to the SD card and boot from it directly.
3. Follow the following steps to prepare an SD/MMC card to boot up an i.MX board using a Linux host machine.
   a. Download latest Real-time Edge release pre-build image for the EVK board from https://www.nxp.com/rtedge.

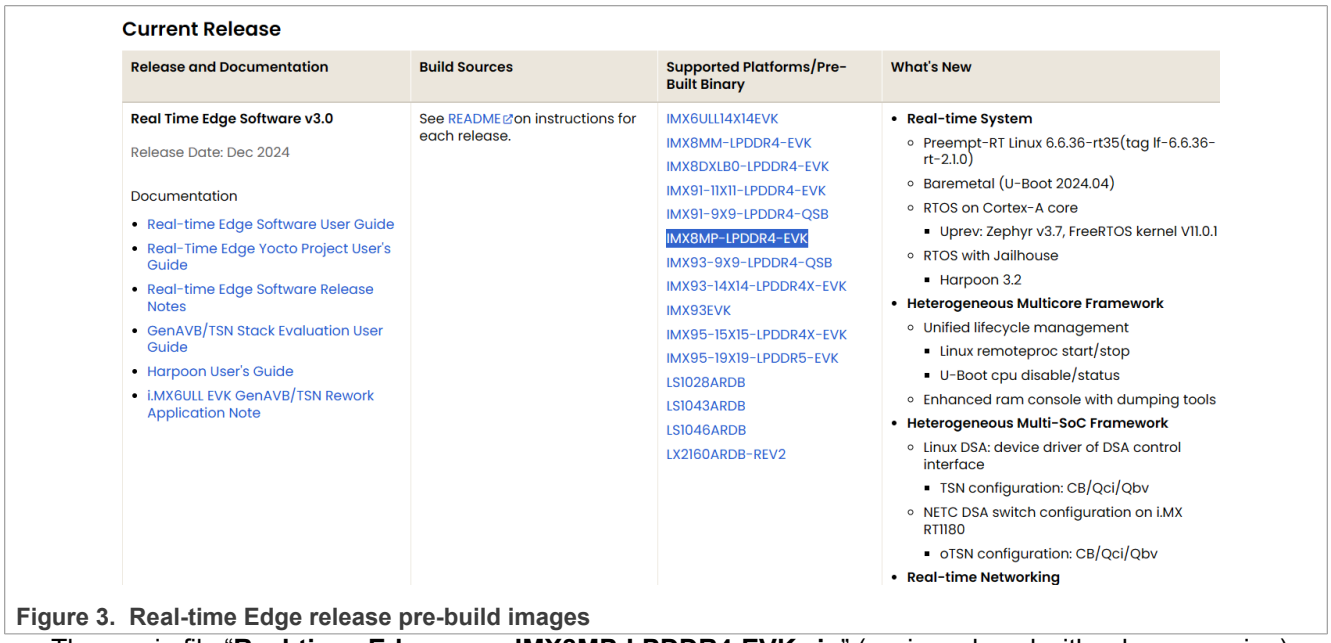

Figure 3. Real-time Edge release pre-build images

Then a zip file "**Real-time_Edge_vx.x_IMX8MP-LPDDR4-EVK.zip**" (x.x is replaced with release version) is downloaded. Extract the `.wic` file named "`nxp-image-real-time-edge-imx8mp-lpddr4-evk.rootfs.wic`" from the path "`Real-time_Edge_vx.x_IMX8MP-LPDDR4-EVK/real-time-edge/nxp-image-real-time-edge-imx8mp-lpddr4-evk.rootfs.wic.zst`" in the zip file.
The SD card image (with the extension .wic) contains U-Boot, the Linux image and device trees, and the rootfs for a 4 GB SD card. The image can be installed on the SD card with one command if flexibility is not required. Carry out the following command to copy the SD card image to the SD/MMC card. Change sdx below to match the one used by the SD card.

```
$ sudo dd if=<image name>.wic of=/dev/sdx bs=1M && sync
```

The entire contents of the SD card are replaced. If the SD card is larger than 4 GB, the additional space is not accessible. Then, insert the SD card back to the EVK board and change the EVK board to boot from SD card by switching SW4[1:4] to be "0011".

4. Power up the board and you can locate the TF-A and U-Boot log from the third port (it is `/dev/ttyUSB2` in the above example).

4.2.5.2 Booting Zephyr by using U-Boot commands

Multiple U-Boot commands can be used to start or stop Zephyr on the Cortex-A core.

4.2.5.2.1 “cpu” command

Following is the help information of the U-Boot `cpu` command:

```
u-boot=> cpu
```

```
cpu - Multiprocessor CPU boot manipulation and release
```

Usage:

- `cpu <num> reset` - Resets cpu <num>
- `cpu status` - Displays the status of all cpus
- `cpu <num> status` - Displays the status of cpu <num>
- `cpu <num> disable` - Disables the status of cpu <num>
- `cpu <num> release <addr> [args]` - Releases cpu <num> at <addr> with [args]

Except the “`cpu <num> reset`” command, all other cpu commands have been implemented on i.MX 8M Mini.

In general, U-Boot runs on the master Core (Core0) of Cortex-A cores. Therefore you can use “`cpu <num> release <addr> [args]`” to start RTOS on any other slave Cortex-A core (the Cores except Core0) from the specified memory address. For this purpose, you must load the RTOS binary images into the corresponding memory space, and use the “`cpu <num> disable`” command to power off any slave core that runs RTOS. Use the commands “`cpu status`” or “cpu <num>” status to check the status (running or power off) for all or the specified Cortex-A cores.

4.2.5.2.2 “go” command

The following is the help information of the U-Boot `go` command:

```
U-Boot=> go
```

go - starts the application at address 'addr'

Usage:

```
go addr [arg ...]
```

- starts the application at address '`addr`'

Passing '`arg`' as arguments

The “`go`” command can be used to start the application or RTOS running on the master Core (Core0). The impact is not returned back to the U-boot command line.

In summary, you can use U-Boot command “`go`” to boot the RTOS from Core0 or use U-Boot command “`cpu`” to boot or power off the RTOS running on the other Cortex-A cores except Core0.

4.2.5.3 Booting Zephyr by using Linux RemoteProc

Remoteproc (Remote Processor Framework) under Linux is another Cortex-A Core Zephyr RTOS management tool under Linux. Apart from the U-Boot commands in U-Boot, using remoteproc can dynamically start or stop Zephyr under Linux. The limitation of this Cortex-A Core Zephyr life cycle management method is that Linux must be running on at least single Cortex-A Core to manage Zephyr running on any of the other Cortex-A Cores.

Remoteproc under Linux is used to control (power on, load firmware, power off) remote processors on which may run different instances of the operating system. For SMP Linux, which it runs on multiple Cortex-A cores,

each Cortex-A core can be used as a remote processor and can use remoteproc to bring up or bring down another RTOS instance on it. To run another RTOS on it, remoteproc firstly removes this Cortex-A Core from SMP Linux by using CPU hotplug and brings it down. Then, it brings up Cortex-A Core again with new RTOS instance to run it. Remoteproc can also bring this CPU Core running RTOS to be down and then plug it back to SMP Linux by using CPU hotplug.

- **Linux Device Tree Configuration**

To use remoteproc to manage life cycle on Cortex-A Core, you must add device nodes in Linux dts. For example, the following dts nodes are defined in imx8m-rproc-ca53.dtsi (provided in Real-time Edge Linux kernel: https://github.com/nxp-real-time-edge-sw/real-time-edge-linux) which defines four remoteproc instances, it use fsl,cpus-bits defines CPU core bitmask to specify which CPU core(s) can be managed by this instance, and memory-region is used to specify the reserved memory space used by RTOS to be run on this instance.

```
        ca53_1: remoteproc-ca53-1 {
                compatible = "fsl,imx-rproc-psci";
                /* bitmask:0b0010, assign A53 Core 1 */
                fsl,cpus-bits = <0x2>;
                memory-region = <&rtos_ca53_reserved>;
        };
        ca53_2: remoteproc-ca53-2 {
                compatible = "fsl,imx-rproc-psci";
                /* bitmask:0b0100, assign A53 Core 2 */
                fsl,cpus-bits = <0x4>;
                memory-region = <&rtos_ca53_reserved>;
        };
        ca53_3: remoteproc-ca53-3 {
                compatible = "fsl,imx-rproc-psci";
                /* bitmask:0b1000, assign A53 Core 3 */
                fsl,cpus-bits = <0x8>;
                memory-region = <&rtos_ca53_reserved>;
        };
        ca53_2_3: remoteproc-ca53-2-3 {
                compatible = "fsl,imx-rproc-psci";
                /* bitmask:0b1100, assign A53 Core 2 and Core 3 */
                fsl,cpus-bits = <0xc>;
                memory-region = <&rtos_ca53_reserved>;
        };
```

The fourth remoteproc instance “ca53_2_3” uses two CPU Core: Core2 and Core3, so it can used to run a SMP RTOS on these two CPU Cores. As remoteproc instance “ca53_3” and “ca53_2_3” both use CPU Core Core3, so these two instances can’t run simultaneously.

- **Hardware Resource Allocation between RTOS and Linux**

In order to run a flexible AMP system with Multiple RTOS and Linux running together on the single SoC simultaneously, the hardware resources need to be allocated to different OS carefully to avoid conflicts, the hardware resources include memory and peripherals. Refer to “imx8mp-evk-multicore-rtos.dts” in Real-time Edge Linux (https://github.com/nxp-real-time-edge-sw/real-time-edge-linux) as an example, the memory used by RTOS need to be added to reserved-memory in Linux dts, and Linux dts nodes UART4 should be disabled as it is used as RTOS debug console, .

- **GIC Configuration to run multiple RTOS on Cortex-A Cores** When running multiple OSes on different Cortex-A Cores on one platform, you must avoid GIC to be re-configured to crash the OS that has already been started. For Zephyr, you need to enable "CONFIG_GIC_SAFE_CONFIG=y". For the Linux kernel, you must enable "CONFIG_GIC_GENTLE_CONFIG=y".

- **Using remoteproc to start or stop RTOS on Cortex-A Core** Use the following steps to start or stop RTOS by using remoteproc:

1. **Start Linux** To run RTOS, users must use a device tree that has remoteproc enabled and has compatible resource allocation between RTOS and Linux. In Real-time Edge Linux, “imx8mp-evk-multicore-rtos.dtb” can be used to run the Zephyr hello_world application.
2. To do this, when U-Boot is executing, stop at the U-Boot prompt with a terminal emulator connected to the serial port and execute the following commands:

```
u-boot=> setenv fdtfile imx8mp-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
u-boot=> run bsp_bootcmd
```

3. Using RemoteProc to start or stop Zephyr: When Linux boots up, log in Linux. Then use “scp” or other method to download or copy zephyr.elf to directory: /lib/firmware/
   The following remoteproc sys portals are available at the following directories:

```
/sys/devices/platform/remoteproc-ca53-1/remoteproc/remoteproc0/
/sys/devices/platform/remoteproc-ca53-2/remoteproc/remoteproc1/
/sys/devices/platform/remoteproc-ca53-3/remoteproc/remoteproc2/
/sys/devices/platform/remoteproc-ca53-2-3/remoteproc/remoteproc3/
```

Note: *remoteproc portal instance may change according to the different device tree configuration used. Therefore, use the name under each remoteproc instance portal to check which instance needs to be used.*

Then, use the following command to start Zephyr:

```
root@imx8mp-lpddr4-evk:~# echo zephyr.elf > /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/firmware
root@imx8mp-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-ca53-3/
remoteproc/remoteproc2/state
```

Then, the Zephyr log is displayed on the Zephyr UART console.

If needed, use the following command to stop RTOS running on Core1:

```
root@imx8mp-lpddr4-evk:~# echo stop > /sys/devices/platform/remoteproc-ca53-3/
remoteproc/remoteproc2/state
```

- **Cortex-A Core dynamically allocated between Linux and RTOS**

Before running RTOS by using remoteproc, all Cortex-A Cores are used by SMP Linux kernel by default. After running RTOS on the specified Cortex-A Core by using remoteproc, these Cortex-A core(s) will be hot removed from SMP Linux kernel and run RTOS on them. After using the remoteproc stop command to stop the RTOS running on Cortex-A Core, these CPU Cores once used by RTOS will be hot added back to SMP Linux kernel. Can use the following command to check which CPU Core are used by SMP Linux kernel currently:

```
root@imx8mp-lpddr4-evk:~# cat /proc/cpuinfo
```

4.3 i.MX 8M Mini

4.3.1 Overview

NXP's i.MX 8M Mini applications processor demonstrates the latest video and audio experience by combining state-of-the-art media-specific features with high-performance processing while keeping the design optimized for lowest power consumption.

The i.MX 8M Mini family of processors features advanced implementation of a quad Arm Cortex-A53 core, which operates at speeds of up to 1.6 GHz. A general-purpose Cortex-M4 400 MHz core processor is used for low-power processing. The DRAM controller supports 32-bit/16-bit LPDDR4, DDR4, and DDR3L memories. A wide range of audio interfaces are available, including I2S, AC97, TDM, and S/PDIF. There are a number of other interfaces for connecting peripherals, such as USB, PCIe, and Ethernet.

Refer to NXP Official website for more details: i.MX 8M Mini | Arm Cortex A53 | Cortex M4 | NXP Semiconductors

4.3.2 Zephyr SoC and Board

From Zephyr 3.7 release, it changed from hardware model v1 (HWMV1) to hardware model v2(HWMV2).

Table 8. HWMV2

Parameter	Path
SoC source code directory	soc/nxp/imx/imx8m/
Board source code directory	boards/nxp/imx8mm_evk/
Supported board name and variants	imx8mm_evk/mimx8mm6/a53 imx8mm_evk/mimx8mm6/a53/smp

Table 9. HWMV1

Parameter	Path
SoC source code directory	soc/arm64/nxp_imx/mimx8m/
Board source code directory	boards/arm64/mimx8mm_evk/
Supported board name and variants	• mimx8mm_evk_a53 • mimx8mm_evk_a53_smp

4.3.3 Peripheral drivers

Table 10. Peripheral drivers on i.MX 8M Mini

Peripheral	Driver	Reference
GIC	GIC v3	5.3 Interrupt-Controller - GIC
Clock	CCM	5.6 Clock – CCM
IOMUX Controller	Pinctrl IMX	5.4 IOMUX – pinctrl_imx
UART	IUART	5.1 UART - IUART
GPIO	IGPIO	5.12 GPIO - IGPIO
I2C	II2C	5.13 I2C - II2C
TPM Counter	GPT	5.9 Counter - GPT

Table 10. Peripheral drivers on i.MX 8M Mini*...continued*

Peripheral	Driver	Reference
Ethernet - ENET	ENET	5.15 Ethernet - ENET

4.3.4 Building the Zephyr application on i.MX 8M Mini

Take the `samples/hello_world` application as an example.

HWMV2:

```
west build -p always -b imx8mm_evk/mimx8mm6/a53 samples/hello_world/
```

HWMV1:

```
west build -p always -b mimx8mm_evk_a53 samples/hello_world/
```

Then, Zephyr binary image "`zephyr.bin`" and elf image "`zephyr.elf`" can be located in the `zephyr/build/zephyr/` directory.

4.3.5 Running Zephyr on i.MX 8M Mini EVK board

Zephyr on Cortex-A core can be booted up or stopped on a specified Cortex-A core by using the U-Boot commands or remoteproc under Linux.

4.3.5.1 Board setup and software preparation (i.MX 8M Mini EVK)

Prepare the i.MX 8M Mini EVK board to run Zephyr:

1. **UART console setup** Connect USB debug connector J901 to PC, PC enumerates two COM ports when the USB cable is plugged into J901. For example on Linux host, the two COM ports could be /dev/ttyUSB0 ~ /dev/ttyUSB1, the ID of ttyUSB device could be changed if multiple USB serial cables are pluged into the same PC, so could check the kernel log after i.MX EVK debug cable is pluged into the PC, for example, the following log shows the COM ports should be /dev/ttyUSB0 ~ /dev/ttyUSB1.

```
linux:~$ sudo dmesg | tail
[525744.711621] usb 3-2.2: New USB device found, idVendor=0403,
 idProduct=6010, bcdDevice= 5.00
[525744.711640] usb 3-2.2: New USB device strings: Mfr=1, Product=2,
 SerialNumber=0
[525744.711646] usb 3-2.2: Product: Dual RS232
[525744.711652] usb 3-2.2: Manufacturer: FTDI
[525744.723483] ftdi_sio 3-2.2:1.0: FTDI USB Serial Device converter detected
[525744.723590] usb 3-2.2: Detected FT2232C/D
[525744.724076] usb 3-2.2: FTDI USB Serial Device converter now attached to
 ttyUSB0
[525744.724621] ftdi_sio 3-2.2:1.1: FTDI USB Serial Device converter detected
[525744.724705] usb 3-2.2: Detected FT2232C/D
[525744.725187] usb 3-2.2: FTDI USB Serial Device converter now attached to
 ttyUSB1
```

   The second COM port (it is /dev/ttyUSB1 in above example) is used for U-Boot and Linux, and the first COM port (it is /dev/ttyUSB0 in above example) is used for Zephyr debug console by default. Developers can use minicom, Putty, Tera Term, Xshell, or other terminal tools to open these two UART consoles with the 115200 Baud rate, 8bit data bits, 1 stop bits and none parity.
2. **Deploying the pre-built SD card image** Zephyr on Cortex-A Core can be booted up or stopped on specified Cortex-A Core by using the U-Boot commands or remoteproc under Linux. U-Boot commands and remoteproc to start or stop Cortex-A Core Zephyr are supported in Real-time Edge software release in v3.0

and late releases (https://www.nxp.com/rtedge), so a quick method to setup software environment on i.MX 8M Mini EVK is to deploy the pre-build image (.wic image) to SD card and boot from it directly.

3. Follow the following steps to prepare an SD/MMC card to boot up an i.MX board using a Linux host machine.
4. Download latest Real-time Edge release pre-build image for the EVK board from https://www.nxp.com/rtedge

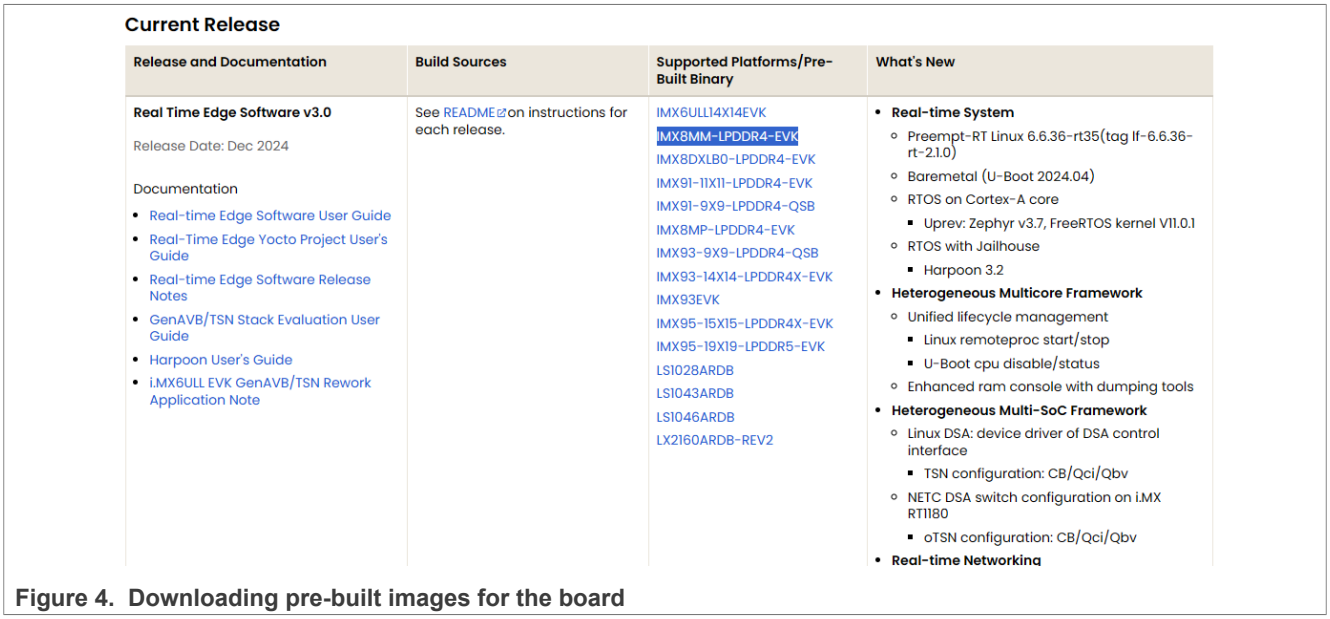

Figure 4. Downloading pre-built images for the board

5. Then a zip file "`Real-time_Edge_vx.x_IMX8MM-LPDDR4-EVK.zip`" (x.x is replaced with release version) is downloaded. Extract the `.wic` file named "`nxp-image-real-time-edge-imx8mm-lpddr4-evk.rootfs.wic`" from the path "`Real-time_Edge_vx.x_IMX8MM-LPDDR4-EVK/real-time-edge/nxp-image-real-time-edge-imx8mm-lpddr4-evk.rootfs.wic.zst`" in the zip file.
6. The SD card image (with the extension `.wic`) contains U-Boot, the Linux image and device trees, and the rootfs for a 4 GB SD card. The image can be installed on the SD card with one command if flexibility is not required. Run the following command to copy the SD card image to the SD/MMC card. Change `sdx` in the below command to match the one used by the SD card.

```
$ sudo dd if=<image name>.wic of=/dev/sdx bs=1M && sync
```

The entire contents of the SD card are replaced. If the SD card is larger than 4 GB, the additional space is not accessible.

7. Then insert SD card back to the EVK board, and change the EVK board to boot from SD card by switching SW4[1:4] to be "0011". Power up the board and then you can locate the TF-A and U-Boot log from the third port (it is /dev/ttyUSB2 in the above example).

4.3.5.2 Running Zephyr application on i.MX 8M Mini EVK

Use the steps below to run the Zephyr application on the i.MX 8M Mini EVK board.

1. Power up the i.MX 8M Mini EVK board and stop at the U-Boot command line.
2. Deploy the Zephyr binary image "`zephyr.bin`" to the board. There are multiple methods that can be used to deploy zephyr.bin to the board. Select one of the below methods:
   - Download the image into memory from the tftp server:

```
u-boot=> tftp 0x93c00000 zephyr.bin
```

- Or copy the image to SD card by using a Linux host PC.

3. There are two partitions in the SD Card if .wic is deployed on the SD Card. The first one is the FAT partition, which is used to store the Linux Image and dtb files. The second partition is the Linux partition to be uses for Linux Root file system.
   - Copy the Zephyr binary image "`zephyr.bin`" to the first FAT partition of the SD card by using Linux host PC and then plug the card into the board. Then, power up the board and stop at U-Boot command line. After that, load the image from SD card into the memory by using the following U-Boot command:

     ```
     u-boot=> fatload mmc 1:1 0x93c00000 zephyr.bin;
     ```

   - Or download the image into memory by using serial port For example, load binary file over serial line in ymodem mode, execute the following command under U-Boot:

     ```
     u-boot=> loady 0x93c00000
     ```

     Then, send the binary file from console software in host PC, different steps for different console software, here take minicom as an example: Use the shortcut keys "ctrl + A + Z" (pressed down and hold "ctrl" key and then press the key "A" and "Z" in turn) to open command windows:

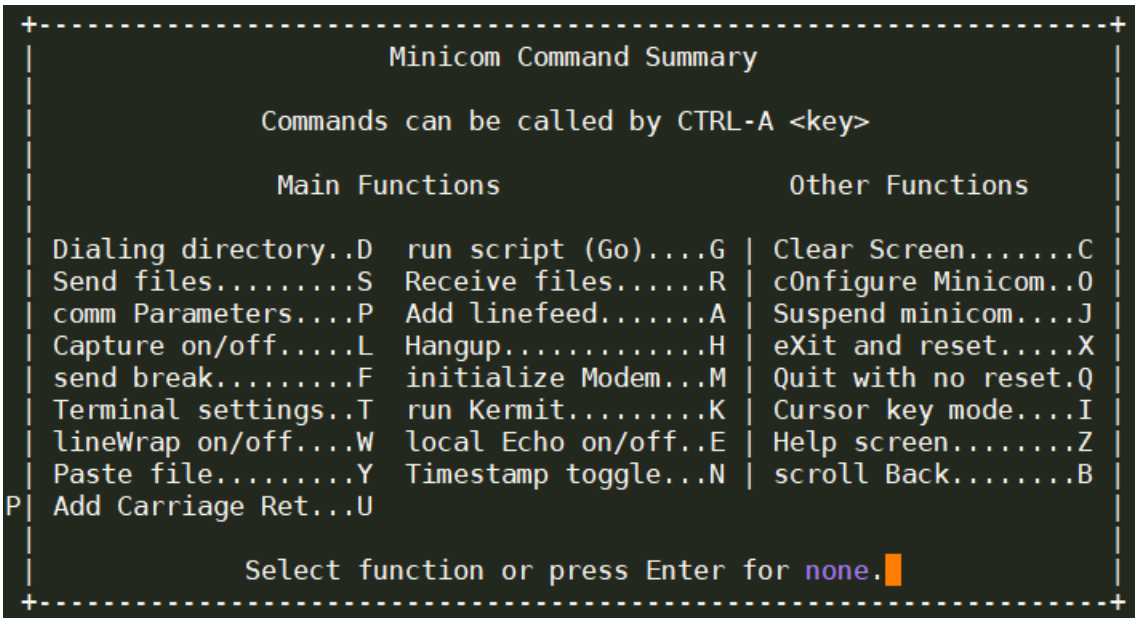

Figure 5. Minicom Command Summary

- Then press "S" to select "Send Files" command:

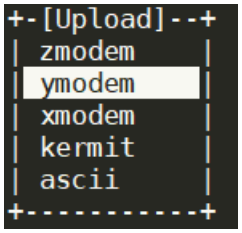

Figure 6. Select ymodem

- Select ymodem, then select or input the path and name of zephyr.bin

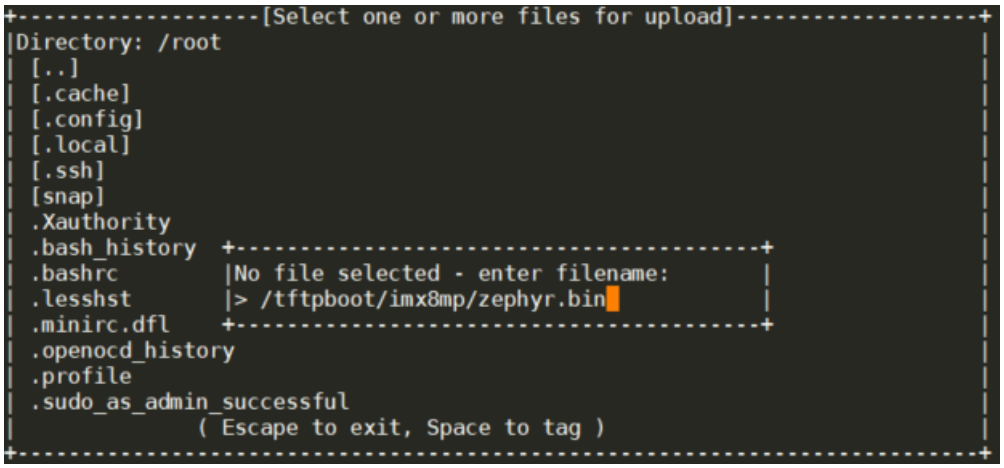

Figure 7. Select or input the path and name of zephyr.bin

- Then zephyr.bin will be download to the memory by through the serial port:

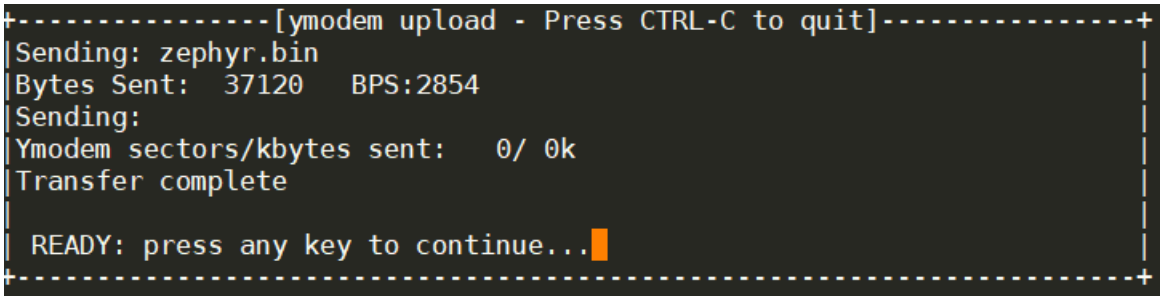

Figure 8. Download the zephyr.bin file

4. Then kick zephyr.bin to Cortex-A53 Core3:

```
u-boot=> dcache flush; icache flush; cpu 3 release 0x93c00000
```

- Or kick zephyr.bin to Cortex-A53 Core0:

```
u-boot=> dcache flush; icache flush; go 0x93c00000
```

5. The UART console of Zephyr now displays the Zephyr booting log.
6. If needed, use the following command to power off the RTOS running on Core3 (the first Core is Core0):

```
u-boot=> cpu 3 disable
```

4.3.5.3 Booting Zephyr by using U-Boot commands

Multiple U-Boot commands can be used to start or stop Zephyr on the Cortex-A core.

4.3.5.3.1 “cpu” command

Following is the help information of the U-Boot `cpu` command:

```
u-boot=> cpu
```

```
cpu - Multiprocessor CPU boot manipulation and release
```

Usage:

- `cpu <num> reset` - Resets cpu <num>
- `cpu status` - Displays the status of all cpus
- `cpu <num> status` - Displays the status of cpu <num>
- `cpu <num> disable` - Disables the status of cpu <num>
- `cpu <num> release <addr> [args]` - Releases cpu <num> at <addr> with [args]

Except the "`cpu <num> reset`" command, all other cpu commands have been implemented on i.MX 8M Mini.

In general, U-Boot runs on the master Core (Core0) of Cortex-A cores. Therefore you can use "`cpu <num> release <addr> [args]`" to start RTOS on any other slave Cortex-A core (the Cores except Core0) from the specified memory address. For this purpose, you must load the RTOS binary images into the corresponding memory space, and use the "`cpu <num> disable`" command to power off any slave core that runs RTOS. Use the commands "`cpu status`" or "cpu <num>" status to check the status (running or power off) for all or the specified Cortex-A cores.

4.3.5.3.2 "go" command

The following is the help information of the U-Boot `go` command:

```
U-Boot=> go
```

go - starts the application at address 'addr'

Usage:

```
go addr [arg ...]
```

- starts the application at address '`addr`'

Passing '`arg`' as arguments

The "`go`" command can be used to start the application or RTOS running on the master Core (Core0). The impact is not returned back to the U-boot command line.

In summary, you can use U-Boot command "`go`" to boot the RTOS from Core0 or use U-Boot command "`cpu`" to boot or power off the RTOS running on the other Cortex-A cores except Core0.

4.3.5.3.3 Running Zephyr application on i.MX 8M Mini EVK

Use the steps below to run the Zephyr application on the i.MX 8M Mini EVK board.

1. Power up the i.MX 8M Mini EVK board and stop at the U-Boot command line.
2. Deploy the Zephyr binary image "`zephyr.bin`" to the board. There are multiple methods that can be used to deploy zephyr.bin to the board. Select one of the below methods:
   - Download the image into memory from the tftp server:

     ```
     u-boot=> tftp 0x93c00000 zephyr.bin
     ```

   - Or copy the image to SD card by using a Linux host PC.
3. There are two partitions in the SD Card if .wic is deployed on the SD Card. The first one is the FAT partition, which is used to store the Linux Image and dtb files. The second partition is the Linux partition to be uses for Linux Root file system.
   - Copy the Zephyr binary image "`zephyr.bin`" to the first FAT partition of the SD card by using Linux host PC and then plug the card into the board. Then, power up the board and stop at U-Boot command line. After that, load the image from SD card into the memory by using the following U-Boot command:

     ```
     u-boot=> fatload mmc 1:1 0x93c00000 zephyr.bin;
     ```

- Or download the image into memory by using serial port For example, load binary file over serial line in ymodem mode, execute the following command under U-Boot:

```
u-boot=> loady 0x93c00000
```

Then, send the binary file from console software in host PC, different steps for different console software, here take minicom as an example: Use the shortcut keys “ctrl + A + Z” (pressed down and hold “ctrl” key and then press the key “A” and “Z” in turn) to open command windows:

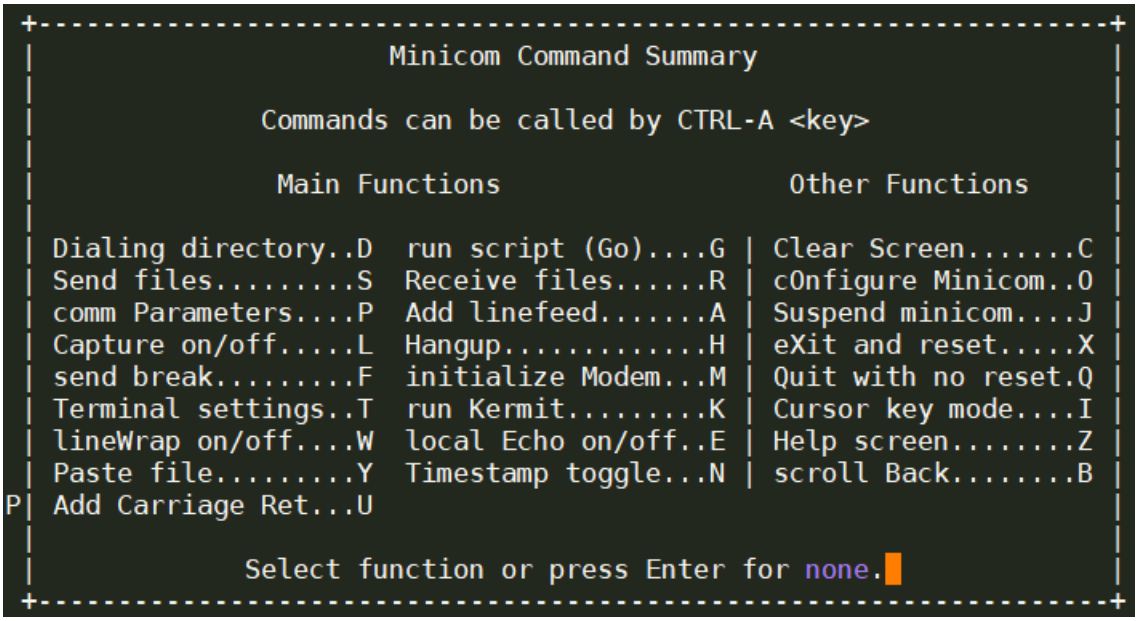

Figure 9. Minicom Command Summary

- Then press “S” to select “Send Files” command:

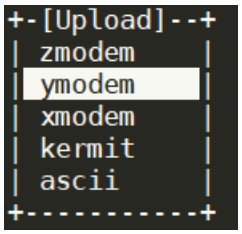

Figure 10. Select ymodem

- Select ymodem, then select or input the path and name of zephyr.bin

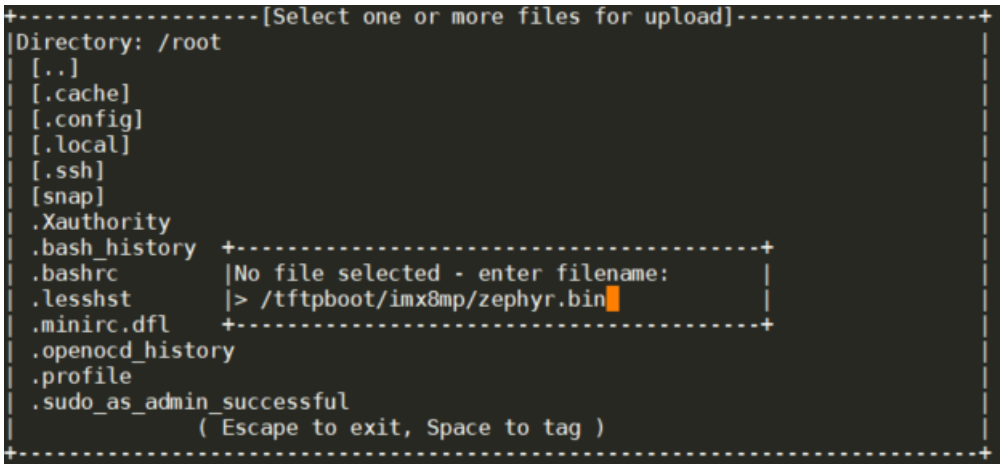

Figure 11. Select or input the path and name of zephyr.bin

- Then zephyr.bin will be download to the memory by through the serial port:

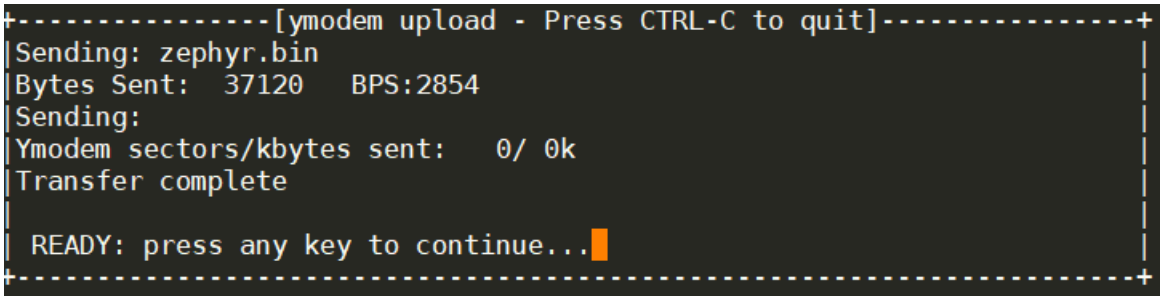

Figure 12. Download the zephyr.bin file

4. Then kick zephyr.bin to Cortex-A53 Core3:

```
u-boot=> dcache flush; icache flush; cpu 3 release 0x93c00000
```

- Or kick zephyr.bin to Cortex-A53 Core0:

```
u-boot=> dcache flush; icache flush; go 0x93c00000
```

5. The UART console of Zephyr now displays the Zephyr booting log.
6. If needed, use the following command to power off the RTOS running on Core3 (the first Core is Core0):

```
u-boot=> cpu 3 disable
```

4.3.5.4 Booting Zephyr by using Linux RemoteProc

Remoteproc (Remote Processor Framework) under Linux is another Cortex-A core Zephyr RTOS management tool in addition to the U-Boot commands in U-Boot. Using remoteproc can dynamically start or stop Zephyr under Linux. The limitation of this Cortex-A Core Zephyr life cycle management method is that Linux must run on at least single Cortex-A Core in order to manage Zephyr running on any of the other Cortex-A Cores.

Remoteproc (Remote Processor Framework) under Linux is used to control (power on, load firmware, power off) remote processors on which may run different instances of operation system. For SMP Linux which it runs on multiple Cortex-A Cores, each Cortex-A Core can be used as a remote processor, and can use remoteproc to bring up or bring down another RTOS instance on it. In order to run another RTOS on it, remoteproc first remove this Cortex-A Core from SMP Linux by using CPU hotplug, and then brings it to be down. Then brings it up again with new RTOS instance to run it. Remoteproc can also bring this CPU Core running RTOS to be down and then plug it back to SMP Linux by using a CPU hotplug.

- **Linux Device Tree Configuration**
  In order to use remoteproc to manage life cycle on Cortex-A Core, you must add device nodes in Linux dts. For example, the following dts nodes are defined in imx8m-rproc-ca53.dtsi (provided in Real-time Edge Linux kernel: https://github.com/nxp-real-time-edge-sw/real-time-edge-linux) which defines four remoteproc instances, it use fsl, cpus-bits defines CPU Core bitmask to specify which CPU Core(s) can be managed by this instance, and memory-region is used to specify the reserved memory space used by RTOS to be run on this instance.

```
ca53_1: remoteproc-ca53-1 {
    compatible = "fsl,imx-rproc-psci"; /* bitmask:0b0010, assign A53 Core 1 */
    fsl,cpus-bits = <0x2>;
    memory-region = <&rtos_ca53_reserved>;
};
ca53_2: remoteproc-ca53-2 {
    compatible = "fsl,imx-rproc-psci"; /* bitmask:0b0100, assign A53 Core 2 */
    fsl,cpus-bits = <0x4>;
    memory-region = <&rtos_ca53_reserved>;
};
ca53_3: remoteproc-ca53-3 {
    compatible = "fsl,imx-rproc-psci"; /* bitmask:0b1000, assign A53 Core 3 */
    fsl,cpus-bits = <0x8>;
    memory-region = <&rtos_ca53_reserved>;
};
ca53_2_3: remoteproc-ca53-2-3 {
    compatible = "fsl,imx-rproc-psci"; /* bitmask:0b1100, assign A53 Core 2 and
 Core 3 */
    fsl,cpus-bits = <0xc>;
    memory-region = <&rtos_ca53_reserved>;
};
```

  The fourth remoteproc instance “ca53_2_3” uses two CPU Cores: Core2 and Core3, so it can used to run a SMP RTOS on these two CPU cores. The remoteproc instances “ca53_3” and “ca53_2_3” both use CPU core Core3, so these two instances cannot run simultaneously.

- **Hardware Resource Allocation between RTOS and Linux**
  In order to run a flexible AMP system with Multiple RTOS and Linux running together on the single SoC simultaneously, the hardware resources must be allocated to different OSes carefully to avoid conflicts. These hardware resources include the memory and peripherals. Refer to “`imx8mp-evk-multicore-rtos.dts`” in Real-time Edge Linux (https://github.com/nxp-real-time-edge-sw/real-time-edge-linux) as an example. The memory used by RTOS must be added to the reserved-memory in Linux dts. In addition, the Linux dts nodes UART4 should be disabled as it is used as the RTOS debug console.

- **GIC Configuration to run multiple RTOS on Cortex-A Cores** When running multiple OSes on different Cortex-A Cores on one platform, you must avoid GIC to be re-configured to crash the OS that has already been started. to achieve this step configure the following:
  – For Zephyr, enable "`CONFIG_GIC_SAFE_CONFIG=y`".
  – For the Linux kernel, enable "`CONFIG_GIC_GENTLE_CONFIG=y`".

- **Using remoteproc to start or stop RTOS on Cortex-A Core**: Use the following steps to start or stop RTOS by using remoteproc:
  1. **Start Linux** In order to run RTOS, must use a device tree which is enabled remoteproc and has compatible resource allocation between RTOS and Linux. In Real-time Edge Linux, “imx8mm-evk-multicore-rtos.dtb” can be used to run Zephyr hello_world application. To do this, when U-Boot is executing, stop at the U-Boot prompt with a terminal emulator connected to the serial port and execute the following commands:

```
u-boot=> setenv fdtfile imx8mm-evk-multicore-rtos.dtb
u-boot=> setenv mmcargs $mmcargs clk_ignore_unused
```

```
u-boot=> run bsp_bootcmd
```

2. **Using RemoteProc to start or stop Zephyr** When Linux boots up, log in Linux. Then use “scp” or other method to download or copy zephyr.elf to directory: /lib/firmware/. The following remoteproc sys portals are available at the following directories:

```
/sys/devices/platform/remoteproc-ca53-1/remoteproc/remoteproc0/
/sys/devices/platform/remoteproc-ca53-2/remoteproc/remoteproc1/
/sys/devices/platform/remoteproc-ca53-3/remoteproc/remoteproc2/
/sys/devices/platform/remoteproc-ca53-2-3/remoteproc/remoteproc3/
```

Note: *The remoteproc portal instance may change according to the different device tree configuration used. Therefore, use the name under each remoteproc instance portal to check which instance needs to be used.*

3. ***Note:*** *Then use the following command to start Zephyr:*

```
root@imx8mm-lpddr4-evk:~# echo zephyr.elf > /sys/devices/platform/
remoteproc-ca53-3/remoteproc/remoteproc2/firmware
root@imx8mm-lpddr4-evk:~# echo start > /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/state
```

Zephyr log will display on the Zephyr UART console. If needed, use the following command to stop RTOS running on Core1:

```
root@imx8mm-lpddr4-evk:~# echo stop > /sys/devices/platform/remoteproc-
ca53-3/remoteproc/remoteproc2/stat
```

4. **Cortex-A Core dynamically allocated between Linux and RTOS** Before running RTOS by using remoteproc, all Cortex-A Cores are used by SMP Linux kernel by default. After running RTOS on the specified Cortex-A Core by using remoteproc, these Cortex-A core(s) will be hot removed from SMP Linux kernel and run RTOS on them. After using the remoteproc stop command to stop the RTOS running on Cortex-A Core, these CPU Cores once used by RTOS will be hot added back to SMP Linux kernel. Can use the following command to check which CPU Core are used by SMP Linux kernel currently:

```
root@imx8mm-lpddr4-evk:~# cat /proc/cpuinfo
```

4.4 i.MX 8M Nano

4.4.1 Overview

The i.MX 8M Nano family of applications processors provided by NXP feature cost-effective integration and affordable performance. Featuring up to four Arm™ Cortex™-A53 cores and a single Cortex-M7 core, this family of processors are optimized for smart, connected, power-efficient devices that require graphics, vision, voice control, intelligent sensing, and general-purpose processing.

This family of applications processors are pin-compatible and scalable to the popular i.MX 8M Mini applications processors. These processors are qualified for commercial and industrial level use and backed by the NXP-supported product longevity program. The i.MX 8M Nano processors can be used in any general purpose industrial and IoT application.

Refer to the NXP official website for more details: i.MX 8M Nano| Arm Cortex A53 | Cortex M7 | NXP Semiconductors

4.4.2 Zephyr SoC and Board

From Zephyr 3.7 release, The hardware model v1 (HWMV1) is updated to hardware model v2 (HWMV2).

HWMV2

Table 11. Hardware model v2 (HWMV2)

Parameter	Path
SoC source code directory	soc/nxp/imx/imx8m/
Board source code directory	boards/nxp/imx8mn_evk/
Supported board name and variants	imx8mn_evk/mimx8mn6/a53 imx8mn_evk/mimx8mn6/a53/smp

HWMV1

Table 12. Hardware model v1 (HWMV1)

Parameter	Path
SoC source code directory	soc/arm64/nxp_imx/mimx8m/
Board source code directory	boards/arm64/mimx8mn_evk/
Supported board name and variants	mimx8mn_evk_a53 mimx8mn_evk_a53_smp

4.4.3 Peripheral Drivers

Table 13. Peripheral drivers on i.MX 8M Nano

Peripheral	Driver	Reference
GIC	GIC v3	5.3 Interrupt-Controller - GIC
Clock	CCM	5.6 Clock – CCM
IOMUX Controller	Pinctrl IMX	5.4 IOMUX – pinctrl_imx
UART	IUART	5.1 UART - IUART
GPIO	IGPIO	5.12 GPIO - IGPIO
I2C	II2C	5.13 I2C - II2C

Table 13. Peripheral drivers on i.MX 8M Nano*...continued*

Peripheral	Driver	Reference
TPM Counter	GPT	5.9 Counter - GPT
Ethernet - ENET	ENET	5.15 Ethernet - ENET

4.4.4 Building Zephyr on i.MX 8M Nano

Take the `samples/hello_world application` as an example:

HWMV2:

```
west build -p always -b imx8mn_evk/mimx8mn6/a53 samples/hello_world/
```

HWMV1:

```
west build -p always -b mimx8mn_evk_a53 samples/hello_world/
```

Then, the Zephyr binary image "`zephyr.bin`" and elf image "`zephyr.elf`" can be located in the `zephyr/build/zephyr/` directory.

4.4.5 Running Zephyr on i.MX 8M Nano EVK board

The U-Boot "`cpu`" command is used to load and kick Zephyr to the Cortex-A secondary core. Currently it is supported in the latest U-Boot version by patch serial:

https://patchwork.ozlabs.org/project/uboot/list/?series=417536&archive=both&state=*

Use the steps below for running Zephyr Application on i.MX 8M Nano EVK:

1. UART Console
   Connect USB debug port to the PC. The PC will enumerate two COM ports when the USB cable is plugged into the J901 connector, the first port is used for Linux and second port is for Zephyr debug console by default. Developers can use Putty, Tera Term, Xshell, or other terminal tools to open the UART console with the 115200 Baud rate, 8-bit data bits, 1 stop bit, and none parity.
2. Power up the EVK board and stop at the U-Boot command line.
3. Copy Zephyr binary image "zephyr.bin" to first FAT partition of the SD card and plug the card into the board, then load the image from SD card into memory by using the following U-Boot command:

   ```
   => fatload mmc 1:1 0x93c00000 zephyr.bin;
   ```

   Or download the image into memory from tftp server:

   ```
   => tftp 0x93c00000 zephyr.bin
   ```

4. Then, kick zephyr.bin to Cortex-A53 Core3:

   ```
   => dcache flush; icache flush; cpu 3 release 0x93c00000
   ```

   Or kick zephyr.bin to Cortex-A53 Core0:

   ```
   => dcache flush; icache flush; go 0x93c00000
   ```

Then the Zephyr UART console displays the Zephyr booting log.

4.5 i.MX 95

4.5.1 Overview

The i.MX 95 applications processor family are optimized for safe, secure, power efficient edge computing. These are developed for use in applications that require powerful AI-accelerated vision processing and immersive graphics abilities. These processors feature functional safety, advanced security, and high-performance data processing that are essential for use in aerospace, automotive edge, commercial IoT, industrial, medical, and network platforms.

Refer to the NXP official website for more details: i.MX 95 Applications Processors Family | NXP Semiconductors

4.5.2 Zephyr SoC and Board

Table 14. Zephyr SoC and Board

Parameter	Path
SoC source code directory	soc/nxp/imx/imx9/
Board source code directory	boards/nxp/imx95_evk/
Supported Board name and variants	imx95_evk/mimx9596/a55 imx95_evk/mimx9596/a55/smp

4.5.3 Peripheral drivers

Table 15. Peripheral drivers on i.MX 95

Peripheral	Driver	Reference
GIC	GIC v3	5.3 Interrupt-Controller - GIC
Clock	CCM	5.8 Clock – SCMI Clock
IOMUX Controller	Pinctrl IMX	5.5 IOMUX – SCMI Pinctrl
UART	LPUART	5.2 UART - LPUART

4.5.4 Building Zephyr on i.MX 95

Take the `samples/hello_world` application as an example:

```
west build -p always -b imx95_evk/mimx9596/a55 samples/hello_world/
```

Then, the Zephyr binary image "`zephyr.bin`" and elf image "`zephyr.elf`" can be located in the `zephyr/build/zephyr/` directory.

4.5.5 Booting Zephyr by using U-Boot commands

Multiple U-Boot commands can be used to start or stop Zephyr on the Cortex-A core.

4.5.5.1 “cpu” command

Following is the help information of the U-Boot `cpu` command:

```
u-boot=> cpu
```

```
cpu - Multiprocessor CPU boot manipulation and release
```

Usage:

- `cpu <num> reset` - Resets cpu <num>
- `cpu status` - Displays the status of all cpus
- `cpu <num> status` - Displays the status of cpu <num>
- `cpu <num> disable` - Disables the status of cpu <num>
- `cpu <num> release <addr> [args]` - Releases cpu <num> at <addr> with [args]

Except the “`cpu <num> reset`” command, all other cpu commands have been implemented on i.MX 8M Mini.

In general, U-Boot runs on the master Core (Core0) of Cortex-A cores. Therefore you can use “`cpu <num> release <addr> [args]`” to start RTOS on any other slave Cortex-A core (the Cores except Core0) from the specified memory address. For this purpose, you must load the RTOS binary images into the corresponding memory space, and use the “`cpu <num> disable`” command to power off any slave core that runs RTOS. Use the commands “`cpu status`” or “cpu <num>” status to check the status (running or power off) for all or the specified Cortex-A cores.

4.5.5.2 “go” command

The following is the help information of the U-Boot `go` command:

```
U-Boot=> go
```

go - starts the application at address 'addr'

Usage:

```
go addr [arg ...]
```

- starts the application at address '`addr`'

Passing '`arg`' as arguments

The “`go`” command can be used to start the application or RTOS running on the master Core (Core0). The impact is not returned back to the U-boot command line.

In summary, you can use U-Boot command “`go`” to boot the RTOS from Core0 or use U-Boot command “`cpu`” to boot or power off the RTOS running on the other Cortex-A cores except Core0.

4.5.6 Running Zephyr on i.MX 95

For i.MX 95, Zephyr on the Cortex-A core can be booted up or stopped on the specified Cortex-A core by using the U-Boot commands specified in the following section.

4.5.6.1 Board setup and software preparation (i.MX 95 EVK)

Prepare the i.MX 95 EVK board to run Zephyr using the steps below:

1. **UART Console Setup** Connect USB debug connector J31 to PC, the PC enumerates four COM ports when the USB cable is plugged into J31. For example on Linux host, the four COM ports could be `/dev/ttyUSB0 ~ /dev/ttyUSB3`, the ID of tty USB device could be changed if multiple USB serial cables are plugged into the same PC. Therefore users must check the kernel log after i.MX 95 EVK debug cable is plugged into the PC. For example, the following log shows that the COM ports should be `/dev/ttyUSB0 ~ /dev/ttyUSB3`.

```
linux:~$ sudo dmesg | tail
[272709.527874] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB0
[272709.528196] ftdi_sio 3-2.1.2:1.1: FTDI USB Serial Device converter
 detected
[272709.528266] usb 3-2.1.2: Detected FT4232H
[272709.528582] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
 ttyUSB1
[272709.528888] ftdi_sio 3-2.1.2:1.2: FTDI USB Serial Device converter
 detected
[272709.528957] usb 3-2.1.2: Detected FT4232H
[272709.529217] usb 3-2.1.2: FTDI USB Serial Device converter now attached to
```

   The third port (it is /dev/ttyUSB2 in above example) is used for U-Boot and Linux, and it is also used for Cortex-A55 Zephyr debug console by default. Developers can use minicom, Putty, Tera Term, Xshell, or other terminal tools to open this UART consoles with the 115200 Baud rate, 8bit data bits, 1 stop bits and none parity
2. **Deploying the pre-built SD card image**
   Zephyr on Cortex-A Core can be booted up or stopped on specified Cortex-A Core by using the U-Boot commands. U-Boot commands and remoteproc to start or stop Cortex-A Core Zephyr are supported in Real-time Edge software release v3.0 and late releases (https://www.nxp.com/rtedge), so a quick method to setup software environment on i.MX 95 EVK is to deploy the pre-build image (.wic image) to SD card and boot from it directly.
3. Follow the following steps to prepare an SD/MMC card to boot up an i.MX board using a Linux host machine.
4. Download latest Real-time Edge release pre-build image for the EVK board from https://www.nxp.com/rtedge

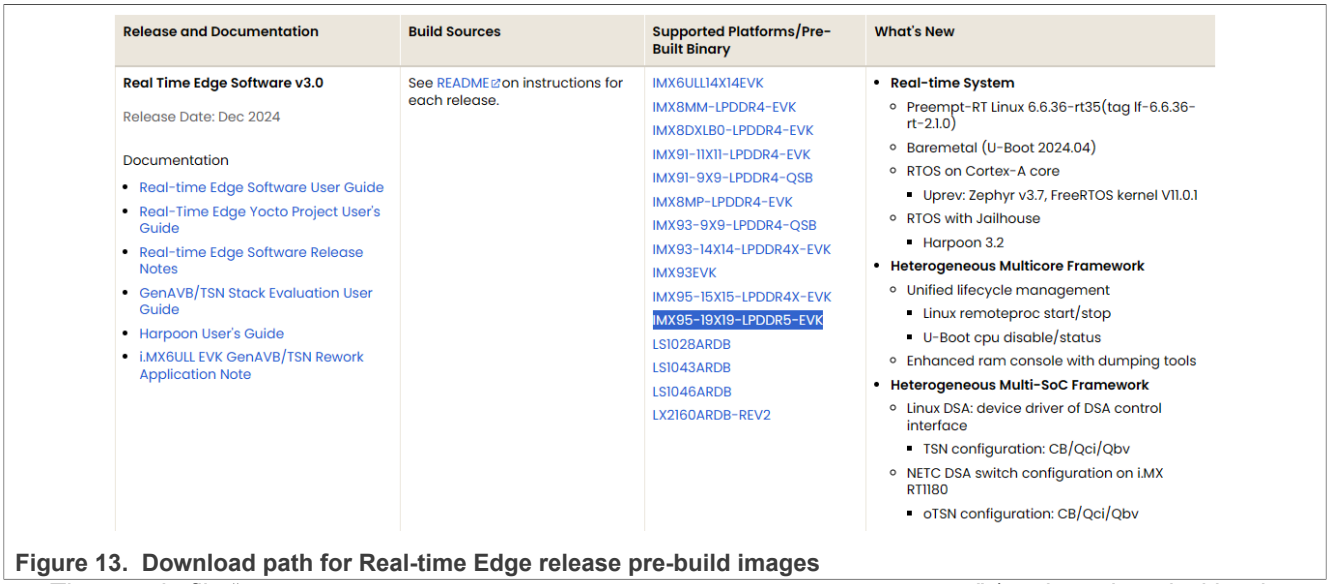

Figure 13. Download path for Real-time Edge release pre-build images

   Then, a zip file "`Real-time_Edge_vx.x_IMX95-19X19-LPDDR5-EVK.zip`" (**x.x** is replaced with release version) is downloaded. Extract the .wic file named "`nxp-image-real-time-edge-imx95-19x19-lpddr5-evk.rootfs.wic`" from the path "`Real-time_Edge_vx.x_IMX95-19X19-LPDDR5-EVK/`

`real-time-edge/nxp-image-real-time-edge-imx95-19x19-lpddr5-evk.rootfs.wic.zst`" in the zip file.

5. The SD card image (with the extension `.wic`) contains U-Boot, the Linux image and device trees, and the rootfs for a 4 GB SD card. The image can be installed on the SD card with one command if flexibility is not required. Carry out the following command to copy the SD card image to the SD/MMC card. Change `sdx` below to match the one used by the SD card.

```
$ sudo dd if=<image name>.wic of=/dev/sdx bs=1M && sync
```

The entire contents of the SD card are replaced. If the SD card is larger than 4 GB, the additional space is not accessible. Then, insert the SD card back to the EVK board. Change the EVK board to boot from SD card by switching SW7[1:4] to be "1011". Power up the board and you can find TF-A and U-Boot log from the third port (it is /dev/ttyUSB2 in the above example).

4.5.6.2 Running the Zephyr Application on i.MX 95 EVK

The below example shows how to run the `hello_world` examples on the i.MX 95 EVK board.

1. Power up the i.MX 95 EVK board and stop at the U-Boot command line.
2. Deploy Zephyr binary image "`zephyr.bin`" to the board.
   There are multiple methods can be used to deploy the `zephyr.bin` to the board, select one the below methods:
3. Download the image into memory from tftp server:

```
u-boot=> tftp 0xd0000000 zephyr.bin
```

4. Or copy the image to SD Card by using Linux host PC
   There are two partitions in the SD Card if .wic is deployed on the SD Card. The first one is the FAT partition, which is used to store Linux image and dtb files. The second one is the Linux partition to be uses for the Linux Root file system.
   Copy the Zephyr binary image "`zephyr.bin`" to the first FAT partition of the SD card by using Linux host PC. Then, plug the card into the board, power up the board, and stop at U-Boot command line. Then, load the image from the SD card into memory by using the following U-Boot command:

```
u-boot=> fatload mmc 1:1 0xd0000000 zephyr.bin;
```

   Or download the image into memory by using serial port
5. For example, load binary file over serial line in ymodem mode, execute the following command under U-Boot:

```
u-boot=> loady 0xd0000000
```

   Then, send the binary file from console software in host PC, different steps for different console software. Here, take minicom as an example: Use the quick key "ctrl + A + Z" (pressed down and hold "ctrl" key and then press the key "A" and "Z" in turn) to open the command window.

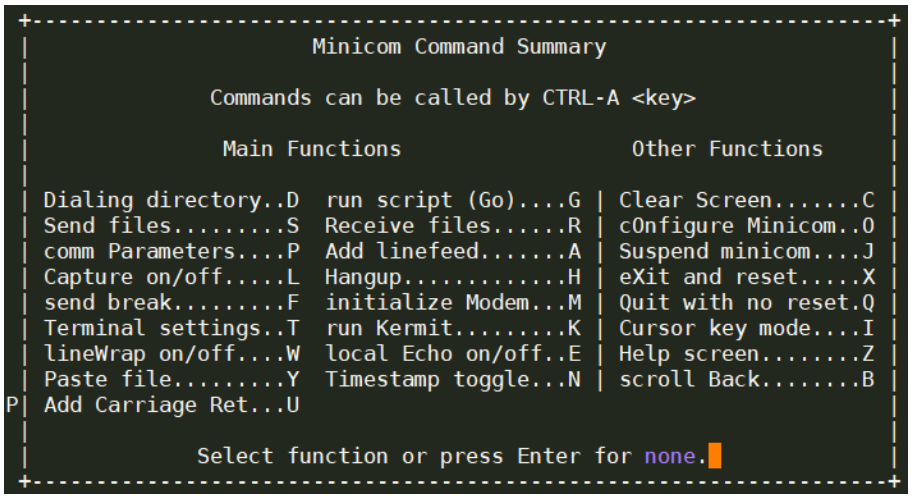

Figure 14. Minicom Command Summary

Then press "**S**" to select the "**Send Files**" command:

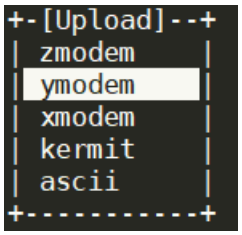

Figure 15. Select ymodem

Select **ymodem**, then select or input the path and name of `zephyr.bin`.

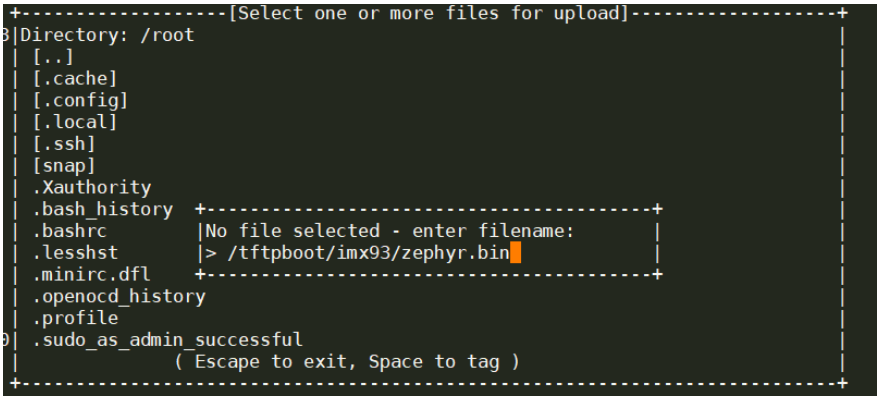

Figure 16. Select or input the path and name of zephyr.bin

Then `zephyr.bin` is downloaded to the memory through the serial port:

Figure 17. Download the zephyr.bin file

6. Then kick `zephyr.bin` to the Cortex-A55 Core1:

```
u-boot=> dcache flush;
icache flush;
cpu 1 release 0xd0000000
```

Or kick the `zephyr.bin` to the Cortex-A55 Core0:

```
u-boot=> dcache flush;
icache flush;
 go 0xd0000000
```

7. The UART console of Zephyr now displays the Zephyr booting log.

5 Zephyr drivers

5.1 UART – IUART

5.1.1 Supported platforms

This driver has been verified on the following platforms:

Table 16. Supported platforms for UART-IUART

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53

5.1.2 Driver information

Table 17. Driver information for UART-IUART

Parameter	Path
Driver source code	`drivers/serial/uart_mcux_iuart.c`
Kconfig	CONFIG_UART_MCUX_IUART
DTS Binding	`dts/bindings/serial/nxp,imx-iuart.yaml`
	compatible: "nxp,imx-iuart"

	Examples:
	`dts/arm64/nxp/nxp_mimx8mp_a53.dtsi`
	`uart2: uart@30890000`
	`{`
	`compatible = "nxp,imx-iuart";`
	`reg = <0x30890000 DT_SIZE_K(64)>;`
	`interrupts = <GIC_SPI 27 IRQ_TYPE_LEVEL`
	`IRQ_DEFAULT_PRIORITY>;`
	`interrupt-names = "irq_0";`
	`nterrupt-parent = <&gic>;`
	`clocks = <&ccm IMX_CCM_UART2_CLK 0x6c 24>;`
	`rdc = <(RDC_DOMAIN_PERM(A53_DOMAIN_ID,`
	`RDC_DOMAIN_PERM_RW)|\`
	`RDC_DOMAIN_PERM(M7_DOMAIN_ID, RDC_DOMAIN_PERM_RW))>;`
	`status = "disabled";`
	`};`

5.2 UART – LPUART

5.2.1 Supported platforms

This driver has been verified on the following platforms:

Table 18. UART-LPUART

Platform	Boards	CPU Core
i.MX 93	i.MX 93 EVK	Cortex-A55

5.2.2 Driver information

Table 19. Driver information for UART-LPUART

Parameter	Path
Driver Source Code	`drivers/serial/uart_mcux_lpuart.c`
Kconfig	`CONFIG_UART_MCUX_LPUART`
DTS Binding	`dts/bindings/serial/nxp,lpuart.yaml`
	compatible: `"nxp,lpuart"`

	Examples:
	`dts/arm64/nxp/nxp_mimx93_a55.dtsi`
	`lpuart1: serial@44380000 {`
	`compatible = "nxp,imx-lpuart", "nxp,lpuart";`
	`reg = <0x44380000 DT_SIZE_K(64)>;`
	`interrupts = <GIC_SPI 19 IRQ_TYPE_LEVEL IRQ_DEFAULT_PRIORITY>;`
	`interrupt-names = "irq_0";`
	`interrupt-parent = <&gic>;`
	`clocks = <&ccm IMX_CCM_LPUART1_CLK 0x6c 24>;`
	`status = "disabled";`
	`};`

5.3 Interrupt Controller – GIC v3

5.3.1 Supported platforms

This driver has been verified on the following platforms:

Table 20. Supported platforms

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53
i.MX 93	i.MX 93 EVK	Cortex-A55

5.3.2 Driver information

Table 21. Driver information for Interrupt Controller – GIC v3

Parameter	Path
Driver Source Code	`drivers/interrupt_controller/intc_gicv3.c`
Kconfig	`CONFIG_GIC_V3`
DTS Binding	`dts/bindings/interrupt-controller/arm,gic-v3.yaml`
	compatible: "`nxp,imx-iuart`"

	Examples:
	`dts/arm64/nxp/nxp_mimx93_a55.dtsi`
	`gic: interrupt-controller@38800000 {`
	`'compatible = "arm,gic-v3", "arm,gic";`
	`reg = <0x38800000 0x10000>, /* GIC Dist */`
	`<0x38880000 0xc0000>; /* GICR (RD_base +`
	`SGI_base) */`
	`interrupt-controller;`
	`#interrupt-cells = <4>;`
	`status = "okay";`
	`};`

5.3.3 Key Notes

CONFIG_GIC_SAFE_CONFIG: Need to enable this configuration item in case multiple OSes running on different CPU Cores which share the same GIC controller, need to avoid the distributor re-configured to avoid crash the OS has already been started. With this enabled, it will bypass GIC distributor configuration if it has been configured by other OS.

5.4 IOMUX – pinctrl_imx

5.4.1 Supported platforms

This driver has been verified on the following platforms:

Table 22. Supported platforms

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53
i.MX 93	i.MX 93 EVK	Cortex-A55

5.4.2 Driver information

Table 23. Driver information for IOMUX – Pinctrl_imx

Parameter	Path
Driver Source Code	`drivers/pinctrl/pinctrl_imx.c`
Kconfig	`CONFIG_PINCTRL_IMX`
DTS Binding	`dts/bindings/pinctrl/nxp,imx-iomuxc.yaml`
	compatible: "`nxp,imx-iomuxc`"

	Examples:
	`dts/arm64/nxp/nxp_mimx93_a55.dtsi`
	`iomuxc: iomuxc@443c0000`
	`{`
	`compatible = "nxp,imx-iomuxc";`
	`reg = <0x443c0000 DT_SIZE_K(64)>;`
	`status = "okay";`
	`pinctrl: pinctrl`
	`{status = "okay";`
	`compatible = "nxp,imx93-pinctrl";`
	`};`
	`};`

5.4.3 Configuring IOMUX on i.MX 93 EVK board

Take LPI2C1 as an example. The pinmux is defined in board dts `boards/nxp/imx93_evk/imx93_evk_mimx9352_a55.dts`:

```
&lpuart1 {
status = "okay";
current-speed = <115200>;
pinctrl-0 = <&lpuart1_default>;
pinctrl-names = "default";
};
```

"`i2c1_default`" is defined in: `boards/nxp/imx93_evk/imx93_evk-pinctrl.dtsi`

```
lpuart1_default: lpuart1_default {
group0 {
```

```
pinmux = <&iomuxc_uart1_rxd_lpuart_rx_lpuart1_rx>,
<&iomuxc_uart1_txd_lpuart_tx_lpuart1_tx>;
bias-pull-up;
slew-rate = "slightly_fast";
drive-strength = "x4";
};
};
```

Group0 includes two pins configuration for scl and sda, these two pins has the same properties, such as driver strength, slew rate etc. These properties are defined in the pinctrl's dts binding file: `dts/bindings/pinctrl/nxp,imx93-pinctrl.yaml`.

IO mux for these two pins "`iomuxc1_i2c1_scl_lpi2c_scl_lpi2c1_scl`" and "`iomuxc1_i2c1_sda_lpi2c_sda_lpi2c1_sda`" are defined in `hal/nxp/dts/nxp/nxp_imx/mimx9352cvuxk-pinctrl.dtsi`, take "`iomuxc1_i2c1_scl_lpi2c_scl_lpi2c1_scl`" as an example:

```
/omit-if-no-ref/ iomuxc_uart1_rxd_lpuart_rx_lpuart1_rx:
 IOMUXC_UART1_RXD_LPUART_RX_LPUART1_RX {
pinmux = <0x443c01d0 0 0x0 0 0x443c03d4>;
};
```

The property array "pinmux = <0x443c0170 0 0x0 0 0x443c0320>" is defined according to definitions in the file: `soc/nxp/imx/imx9/imx93/pinctrl_soc.h`:

```
#define MCUX_IMX_PINMUX(node_id)
mux_register = DT_PROP_BY_IDX(node_id, pinmux, 0), \
.config_register = DT_PROP_BY_IDX(node_id, pinmux, 4), \
.input_register = DT_PROP_BY_IDX(node_id, pinmux, 2), \
.mux_mode = DT_PROP_BY_IDX(node_id, pinmux, 1), \
.input_daisy = DT_PROP_BY_IDX(node_id, pinmux, 3), \
}
```

From the definition, the first value in array is "`0x443c0170`", it is MUX control register address, it is register "SW_MUX_CTL_PAD_I2C1_SCL" according to Chapter IOMUX Controller in i.MX 93 reference manual.

The second value in the array is "0". It is the register value of the MUX control register.From register definition, we can find that it selects MUX mode 0, which selects the pin function as i2c SCL. The third value and the fourth value in the array is "0x0" and "0", they are input register address and register value, that is means no need to set input register. The fifth value in array is 0x443c0320, it is config register address, the value of config register is defined in group0's dts node property.

Refer to the Linux dts configuration for the config register setting value for the specific pin.

5.5 IOMUX – SCMI Pinctrl

5.5.1 Supported platforms

This driver has been verified on the following platforms:

Table 24. Supported platforms

Platform	Boards	CPU Core
i.MX 95	i.MX 95 EVK	Cortex-A55

5.5.2 Driver information

Table 25. Driver information for IOMUX – SCMI Pinctrl

Parameter	Path
Driver Source Code	drivers/pinctrl/pinctrl_imx_scmi.c
Kconfig	CONFIG_PINCTRL_IMX_SCMI
DTS Binding	dts/bindings/firmware/arm,scmi-pinctrl.yaml
	compatible: "arm,scmi-pinctrl "

	Examples:
	`dts/arm64/nxp/nxp_mimx95_a55.dtsi`
	`firmware {`
	`scmi {`
	`compatible = "arm,scmi";`
	`shmem = <&scmi_shmem0>;`
	`mboxes = <&mu2 0>;`
	`mbox-names = "tx";`
	`#address-cells = <1>;`
	`#size-cells = <0>;`
	`scmi_iomuxc: protocol@19 {`
	`compatible = "arm,scmi-pinctrl";`
	`reg = <0x19>;`
	`pinctrl: pinctrl {`
	`compatible = "nxp,imx95-pinctrl", "nxp,imx93-pinctrl";`
	`};`
	`};`
	`};`
	`};`

5.5.3 Configuring IOMUX on i.MX 95 EVK board

Take lpuart1 as an example. The pinmux is defined in board `dts boards/nxp/imx95_evk/mx95_evk_mimx9596_a55.dts`:

```
&lpuart1 {
   status = "okay";
   current-speed = <115200>;
   pinctrl-0 = <&lpuart1_default>;
   pinctrl-names = "default";
 };
```

“lpuart1_default” is defined in: boards/nxp/imx95_evk/imx95_evk-pinctrl.dtsi

```
lpuart1_default: lpuart1_default {
group0 {
pinmux = <&iomuxc_uart1_rxd_lpuart_rx_lpuart1_rx>,
<&iomuxc_uart1_txd_lpuart_tx_lpuart1_tx>;
bias-pull-up;
slew-rate = "slightly_fast";
drive-strength = "x4";
};
};
```

Group0 includes two pins configuration for pin lpuart1_rx and lpuart1_tx, these two pins has the same properties, such as driver strength, slew rate etc, these properties are defined in pinctrl’s dts binding file: dts/bindings/pinctrl/nxp,imx93-pinctrl.yaml (because “pinctrl” node in dts/arm64/nxp/nxp_mimx95_a55.dtsi has the compatible string "nxp,imx93-pinctrl").

IO mux for these two pins “iomuxc_uart1_rxd_lpuart_rx_lpuart1_rx” and “iomuxc_uart1_txd_lpuart_tx_lpuart1_tx” are defined in hal/nxp/dts/nxp/nxp_imx/mimx9596avzxn-pinctrl.dtsi, take “iomuxc_uart1_rxd_lpuart_rx_lpuart1_rx” as example:

/omit-if-no-ref/ iomuxc_uart1_rxd_lpuart_rx_lpuart1_rx: IOMUXC_UART1_RXD_LPUART_RX_LPUART1_RX {

pinmux = <0x443c01d0 0 0x0 0 0x443c03d4>;

};

The property array “pinmux = <0x443c01d0 0 0x0 0 0x443c03d4>” is defined according to definitions in soc/nxp/imx/imx9/imx95/pinctrl_soc.h:

```
#define MCUX_IMX_PINMUX(node_id) \
{
\
.mux_register = DT_PROP_BY_IDX(node_id, pinmux, 0), \
.config_register = DT_PROP_BY_IDX(node_id, pinmux, 4), \
.input_register = DT_PROP_BY_IDX(node_id, pinmux, 2), \
.mux_mode = DT_PROP_BY_IDX(node_id, pinmux, 1), \
.input_daisy = DT_PROP_BY_IDX(node_id, pinmux, 3), \
}
```

From the definition, the first value in array is “0x443c01d0”, it is MUX control register address, it is register “SW_MUX_CTL_PAD_UART1_RXD” according to Chapter *IOMUX Controller* in the i.MX 95 Reference Manual, the second value in array is “0”, it is the register value of MUX control register, from register definition we can find it will select MUX mode 0 which will select the pin function as LPUART1_RX. The third value and the fourth value in array is “0x0” and “0”, they are input register address and register value, that is means no need to set input register. The fifth value in array is 0x443c03d4, it is config register address, the value of config register is defined in group0’s dts node property.

Refer to the Linux dts configuration for the config register setting value for the specific pin.

5.6 Clock – CCM

5.6.1 Supported platforms

This driver has been verified on the following platforms:

Table 26. Clock – CCM supported platforms

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53

5.6.2 Driver information

Table 27. Clock – CCM driver information

Parameter	Path
Driver Source Code	`drivers/clock_control/clock_control_mcux_ccm.c`
Kconfig	`CONFIG_CLOCK_CONTROL_MCUX_CCM`
DTS Binding	`dts/bindings/clock/nxp, imx-ccm.yaml1`
	`compatible: "nxp,imx-ccm"`

	Examples:
	`dts/arm64/nxp/nxp_mimx8mp_a53.dtsi`
	`ccm: ccm@30380000 {`
	`compatible = "nxp,imx-ccm";`
	`reg = <0x30380000 DT_SIZE_K(64)>;`
	`#clock-cells = <3>;`
	`};`

5.7 Clock – CCM Rev2

5.7.1 Supported platforms

This driver has been verified on the following platforms:

Table 28. Supported platforms for CCM Rev2

Platform	Boards	CPU Core
i.MX 93	i.MX 93 EVK	Cortex-A55

5.7.2 Driver information

Table 29. Driver information for CCM Rev2

Parameter	Path
Driver Source Code	`drivers/clock_control/clock_control_mcux_ccm_rev2.c`
Kconfig	`CONFIG_CLOCK_CONTROL_MCUX_CCM_REV2`
DTS Binding	`dts/bindings/clock/nxp, imx-ccm-rev2.yaml`
	compatible: "`nxp,imx-ccm-rev2`"

	Examples:
	`dts/arm64/nxp/nxp_mimx93_a55.dtsi`
	`ccm: ccm@44450000 {`
	`compatible = "nxp,imx-ccm-rev2";`
	`reg = <0x44450000 DT_SIZE_K(64)>;`
	`#clock-cells = <3>;`
	`};`

5.8 Clock – SCMI Clock

5.8.1 Supported platforms

This driver has been verified on the following platforms:

Table 30. Supported platforms

Platform	Boards	CPU Core
i.MX 95	i.MX 95 EVK	Cortex-A55

5.8.2 Driver information

Table 31. Driver information for SCMI clock

Parameter	Path
Driver Source Code	`drivers/clock_control/clock_control_arm_scmi.c`
Kconfig	`CONFIG_CLOCK_CONTROL_ARM_SCMI`
DTS Binding	`dts/bindings/firmware/arm, scmi-clock.yaml`
	compatible: "`arm,scmi-clock`"

	Examples:
	`dts/arm64/nxp/nxp_mimx95_a55.dtsi`
	`firmware {`
	`scmi {`
	`compatible = "arm,scmi";`
	`shmem = <&scmi_shmem0>;`
	`mboxes = <&mu2 0>;`
	`mbox-names = "tx";`
	`#address-cells = <1>;`
	`#size-cells = <0>;`
	`scmi_clk: protocol@14 {`
	`compatible = "arm,scmi-clock";`
	`reg = <0x14>;`
	`#clock-cells = <1>;`
	`};`
	`};`

5.9 Counter - GPT

5.9.1 Supported platforms

This driver has been verified on the following platforms:

Table 32. Supported platforms

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53

5.9.2 Driver information

Table 33. Driver information for Counter- GPT

Parameter	Path
Driver Source Code	`drivers/counter/counter_mcux_gpt.c`
DTS Binding	`dts/bindings/timer/nxp,imx-gpt.yaml`
	compatible: "`nxp,imx-gpt`"

	Examples:
	`dts/arm64/nxp/nxp_mimx8mp_a53.dtsi`
	`gpt1: gpt@302d0000 {`
	`compatible = "nxp,imx-gpt";`
	`reg = <0x302d0000 DT_SIZE_K(64)>;`
	`interrupt-parent = <&gic>;`
	`interrupts = <GIC_SPI 55 IRQ_TYPE_LEVEL IRQ_DEFAULT_PRIORITY>;`
	`gptfreq = <24000000>;`
	`clocks = <&ccm IMX_CCM_GPT_IPG_CLK 0x6C 20>;`
	`status = "disabled";`
	`};`

Kconfig	CONFIG_COUNTER_MCUX_GPT

5.9.3 Driver testing and examples

For driver testing and examples, use the following use case:

- Counter_basic_api (refer to the section Section 6.1)

5.10 Counter - TPM

5.10.1 Supported platforms

This driver has been verified on the following platforms:

Table 34. Supported platforms

Platform	Boards	CPU Core
i.MX 93	i.MX 93 EVK	Cortex-A55

5.10.2 Driver information

Table 35. Driver information for Counter- TPM

Driver Source Code	`drivers/counter/counter_mcux_tpm.c`
Kconfig	`CONFIG_COUNTER_MCUX_TPM`
DTS Binding	`dts/bindings/timer/nxp,tpm-timer.yaml`
	compatible: "`nxp,tpm-timer`"

	Examples:
	`dts/arm64/nxp/nxp_mimx93_a55.dtsi`
	`tpm2: tpm@44320000`
	`{`
	`compatible = "nxp,tpm-timer";`
	`reg = <0x44320000 DT_SIZE_K(64)>;`
	`interrupts = <GIC_SPI 37 IRQ_TYPE_LEVEL IRQ_DEFAULT_PRIORITY>;`
	`interrupt-names = "irq_0";`
	`interrupt-parent = <&gic>;`
	`clocks = <&ccm IMX_CCM_TPM2_CLK 0 0>;`
	`prescaler = <1>;`
	`status = "disabled";`
	`};`

5.10.3 Driver testing and examples

For driver testing and examples, use the following use case:

- Counter_basic_api (refer to the section Section 6.1)

5.11 GPIO - RGPIO

5.11.1 Supported platforms

This driver has been verified on the following platforms:

Table 36. Supported platforms

Platform	Boards	CPU Core
i.MX 93	i.MX 93 EVK	Cortex-A53

5.11.2 Driver information

Table 37. Driver information for GPIO-RGPIO

Parameter	Path
Driver source code	`drivers/gpio/gpio_mcux_rgpio.c`
Kconfig	`CONFIG_GPIO_MCUX_RGPIO`
DTS binding	`dts/bindings/gpio/nxp,imx-rgpio.yaml`
	`compatible: "nxp,imx-rgpio"`

	Examples:
	`dts/arm64/nxp/nxp_mimx93_a55.dtsi`
	`gpio1: gpio@47400000`
	`{compatible = "nxp,imx-rgpio";`
	`reg = <0x47400000 DT_SIZE_K(64)>;`
	`interrupt-parent = <&gic>;`
	`interrupts = <GIC_SPI 10 IRQ_TYPE_LEVEL`
	`IRQ_DEFAULT_PRIORITY>,`
	`<GIC_SPI 11 IRQ_TYPE_LEVEL IRQ_DEFAULT_PRIORITY>;`
	`gpio-controller;`
	`#gpio-cells = <2>;`
	`};`

5.11.3 Driver testing and examples

For driver testing and examples, use the following use cases:

- gpio_basic_api (refer to chapter Section 6.2)
- button and blinky (refer to chapter Section 6.3)

5.12 GPIO - IGPIO

5.12.1 Supported platforms

This driver has been verified on the following platforms:

Table 38. Supported platforms

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53

5.12.2 Driver information

Table 39. Driver information for GPIO-IGPIO

Parameter	Path
Driver Source Code	`drivers/gpio/gpio_mcux_igpio.c`
Kconfig	`CONFIG_GPIO_MCUX_IGPIO`
DTS Binding	`dts/bindings/gpio/nxp,imx-gpio.yaml`
	compatible: "`nxp,imx-gpio`"

	Examples:
	`dts/arm64/nxp/nxp_mimx8mm_a53.dtsi`
	`gpio1: gpio@30200000 {`
	`compatible = "nxp,imx-gpio";`
	`reg = <0x30200000 DT_SIZE_K(64)>;`
	`interrupts = <GIC_SPI 64 IRQ_TYPE_LEVEL`
	`IRQ_DEFAULT_PRIORITY>,`
	`<GIC_SPI 65 IRQ_TYPE_LEVEL IRQ_DEFAULT_PRIORITY>;`
	`interrupt-names = "irq_0", "irq_1";`
	`interrupt-parent = <&gic>;`
	`rdc = <RDC_DOMAIN_PERM(A53_DOMAIN_ID,`
	`RDC_DOMAIN_PERM_RW)>;`
	`gpio-controller;`
	`#gpio-cells = <2>;`
	`ngpios = <30>;`
	`status = "disabled";`
	`};`

5.12.3 Driver testing and examples

For driver testing and examples, use the following use cases:

- Counter_basic_api (refer to the section Section 6.1)

5.13 I2C – II2C

5.13.1 Supported platforms

This driver has been verified on the following platforms:

Table 40. Supported platforms

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53

5.13.2 Driver information

Table 41. Driver information for I2C-II2C

Parameter	Path
Driver Source code	`drivers/i2c/i2c_nxp_ii2c.c`
Kconfig	`CONFIG_I2C_NXP_II2C`
DTS Binding	`dts/bindings/i2c/nxp,ii2c.yaml`
	compatible: "`nxp,ii2c`"

	Examples:
	`dts/arm64/nxp/nxp_mimx8mm_a53.dtsi`
	`i2c1: i2c@30a20000 {`
	`compatible = "nxp,ii2c";`
	`#address-cells = <1>;`
	`#size-cells = <0>;`
	`reg = <0x30a20000 0x10000>;`
	`interrupts = <GIC_SPI 35 IRQ_TYPE_LEVEL`
	`IRQ_DEFAULT_PRIORITY>;`
	`interrupt-parent = <&gic>;`
	`clocks = <&ccm IMX_CCM_I2C1_CLK 0 0>;`
	`rdc = <RDC_DOMAIN_PERM(A53_DOMAIN_ID,`
	`RDC_DOMAIN_PERM_RW)>;`
	`status = "disabled";`
	`};`

5.13.3 Driver testing and examples

For driver testing and examples for i.MX 8M Mini EVK, i.MX 8M Nano EVK, and i.MX 8M plus EVK, use the following use cases:

- gpio_basic_api (refer to the section Section 6.2.
  It uses IO expander by using I2C bus to test GPIO ports.

5.14 I2C – LPI2C

5.14.1 Supported platforms

This driver has been verified on the following platforms:

Table 42. Supported platforms

Platform	Boards	CPU Core
i.MX 93	i.MX 93 EVK	Cortex-A55

5.14.2 Driver information

Table 43. Driver information for I2C -LPI2C

Parameter	Path
Driver Source Code	`drivers/i2c/i2c_mcux_lpi2c.c`
Kconfig	`CONFIG_I2C_MCUX_LPI2C`
DTS Binding	`dts/bindings/i2c/nxp,lpi2c.yaml`
	compatible: "`nxp,lpi2c`"

	Examples:
	`dts/arm64/nxp/nxp_mimx93_a55.dtsi`
	`lpi2c1: i2c@44340000 {`
	`compatible = "nxp,lpi2c";`
	`clock-frequency = <I2C_BITRATE_STANDARD>;`
	`#address-cells = <1>;`
	`#size-cells = <0>;`
	`reg = <0x44340000 0x4000>;`
	`interrupts = <GIC_SPI 13 IRQ_TYPE_LEVEL`
	`IRQ_DEFAULT_PRIORITY>;`
	`interrupt-parent = <&gic>;`
	`clocks = <&ccm IMX_CCM_LPI2C1_CLK 0x70 6>;`
	`status = "disabled";`
	`};`

5.15 Ethernet – ENET

5.15.1 Supported platforms

This driver has been verified on the following platforms:

Table 44. Supported platforms

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53
i.MX 93	i.MX 93 EVK	Cortex-A55

5.15.2 Driver information

Table 45.

Parameter	Path
Driver Source Code	drivers/ethernet/nxp_enet
Kconfig	`CONFIG_ETH_NXP_ENET` `CONFIG_ETH_NXP_ENET_1G`

DTS Binding	`dts/bindings/ethernet/nxp,enet1g.yaml`
	`dts/bindings/ethernet/nxp,enet-mac.yaml`
	`dts/bindings/mdio/nxp,enet-mdio.yaml`

	compatible: `"nxp,enet1g"` `"nxp,enet-mac"` `"nxp,enet-mdio"`
	Examples: `dts/arm64/nxp/nxp_mimx8mp_a53.dtsi`

```
enet: enet@30be0000
{
 compatible = "nxp,enet1g";
 reg = <0x30be0000 DT_SIZE_K(64)>;
 clocks = <&ccm IMX_CCM_ENET_CLK 0 0>;
 rdc = <RDC_DOMAIN_PERM(A53_DOMAIN_ID,
 RDC_DOMAIN_PERM_RW)>;
 status = "disabled";
 enet_mac: ethernet {
  compatible = "nxp,enet-mac";
  interrupts = <GIC_SPI 120 IRQ_TYPE_LEVEL
 IRQ_DEFAULT_PRIORITY>;
  interrupt-names = "COMMON";
  interrupt-parent = <&gic>;
  nxp,mdio = <&enet_mdio>;
  nxp,ptp-clock = <&enet_ptp_clock>;
  status = "disabled";
 };
   enet_mdio: mdio
   {
     compatible = "nxp,enet-mdio";
     #address-cells = <1>;
     #size-cells = <0>;
     status = "disabled";
    };
```

Table 45. *...continued*

Parameter	Path

	enet_ptp_clock: ptp_clock {
	compatible = "nxp,enet-ptp-clock";
	interrupts = <GIC_SPI 121 IRQ_TYPE_LEVEL
	IRQ_DEFAULT_PRIORITY>;
	interrupt-parent = <&gic>;
	clocks = <&ccm IMX_CCM_ENET_PLL 0 0>;
	status = "disabled";
	};
	};

5.15.3 Driver testing and examples

For driver testing and examples, use the following use case: zperf (refer to the Section Section 6.5).

6 Zephyr testing and use cases

6.1 counter_basic_api

6.1.1 Overview

The Zephyr kernel provides the test application "`tests/drivers/counter/counter_basic_api`" that tests the counter subsystem using a specified counter instance.

6.1.2 Supported platforms

Table 46. Supported platforms for counter_basic_api

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53
i.MX 93	i.MX 93 EVK	Cortex-A55

6.1.3 Hardware setup

No extra hardware setup is needed.

6.1.4 Software setup and build

Users must specify the counter instance used by the testing application dta overlay:

- For i.MX 8M Plus EVK:
  - Overlay:

```
tests/drivers/counter/counter_basic_api/boards/
imx8mp_evk_mimx8ml8_a53.overlay
```

  - Building:

```
west build -p always -b imx8mp_evk/mimx8ml8/a53 tests/drivers/counter/
counter_basic_api/
```

- For i.MX 8M Mini EVK:
  - Overlay:

```
tests/drivers/counter/counter_basic_api/boards/
imx8mm_evk_mimx8mm6_a53.overlay
```

  - Building:

```
west build -p always -b imx8mm_evk/mimx8mm6/a53 tests/drivers/counter/
counter_basic_api/
```

- For i.MX 8M Nano EVK:
  - Overlay:

```
tests/drivers/counter/counter_basic_api/boards/
imx8mn_evk_mimx8mn6_a53.overlay
```

  Building:

```
west build -p always -b imx8mn_evk/mimx8mn6/a53 tests/drivers/counter/
counter_basic_api/
```

- For i.MX 93 EVK
  - Overlay:

```
tests/drivers/counter/counter_basic_api/boards/imx93_evk_mimx9352_a55.overlay
```

  - Building:

```
west build -p always -b imx93_evk/mimx9352/a55 tests/drivers/counter/
counter_basic_api/
```

6.1.5 Running the test cases

The following log is produced if all the test cases pass:

```
***** Booting Zephyr OS build v3.6.0-980-g69b472aa6e07 ***
Running TESTSUITE counter_basic
===================================================================
START - test_all_channels
```

```
Testing gpt@302d0000
 PASS - test_all_channels in 0.132 seconds
===================================================================
START - test_cancelled_alarm_does_not_expire
Skipped for gpt@302d0000
 PASS - test_cancelled_alarm_does_not_expire in 0.103 seconds
===================================================================
START - test_late_alarm
Skipped for gpt@302d0000
 PASS - test_late_alarm in 0.103 seconds
===================================================================
START - test_late_alarm_error
Skipped for gpt@302d0000
 PASS - test_late_alarm_error in 0.103 seconds
===================================================================
START - test_multiple_alarms
Skipped for gpt@302d0000
 PASS - test_multiple_alarms in 0.103 seconds
===================================================================
START - test_set_top_value_with_alarm
E: Wrap can only be set to 0xffffffff
Skipped for gpt@302d0000
 PASS - test_set_top_value_with_alarm in 0.106 seconds
===================================================================
START - test_short_relative_alarm
Skipped for gpt@302d0000
 PASS - test_short_relative_alarm in 0.104 seconds
===================================================================
START - test_single_shot_alarm_notop
Testing gpt@302d0000
 PASS - test_single_shot_alarm_notop in 0.172 seconds
===================================================================
START - test_single_shot_alarm_top
E: Wrap can only be set to 0xffffffff
Skipped for gpt@302d0000
 PASS - test_single_shot_alarm_top in 0.106 seconds
===================================================================
TESTSUITE counter_basic succeeded
Running TESTSUITE counter_no_callback
===================================================================
START - test_set_top_value_without_alarm
E: Wrap can only be set to 0xffffffff
Skipped for gpt@302d0000
 PASS - test_set_top_value_without_alarm in 0.106 seconds
===================================================================
TESTSUITE counter_no_callback succeeded
------ TESTSUITE SUMMARY START ------
SUITE PASS - 100.00% [counter_basic]: pass = 9, fail = 0, skip = 0, total = 9
 duration = 1.032 seconds
 - PASS - [counter_basic.test_all_channels] duration = 0.132 seconds
 - PASS - [counter_basic.test_cancelled_alarm_does_not_expire] duration = 0.103
 seconds
 - PASS - [counter_basic.test_late_alarm] duration = 0.103 seconds
 - PASS - [counter_basic.test_late_alarm_error] duration = 0.103 seconds
 - PASS - [counter_basic.test_multiple_alarms] duration = 0.103 seconds
 - PASS - [counter_basic.test_set_top_value_with_alarm] duration = 0.106 seconds
 - PASS - [counter_basic.test_short_relative_alarm] duration = 0.104 seconds
 - PASS - [counter_basic.test_single_shot_alarm_notop] duration = 0.172 seconds
 - PASS - [counter_basic.test_single_shot_alarm_top] duration = 0.106 seconds
```

```
SUITE PASS - 100.00% [counter_no_callback]: pass = 1, fail = 0, skip = 0, total
 = 1 duration = 0.106 seconds
 - PASS - [counter_no_callback.test_set_top_value_without_alarm] duration =
 0.106 seconds
------ TESTSUITE SUMMARY END ------
===================================================================
PROJECT EXECUTION SUCCESSFUL
```

6.2 gpio_basic_api

6.2.1 Overview

The Zephyr kernel provides the test application "`tests/drivers/gpio/gpio_basic_api`". This application tests the GPIO subsystem using a hardware configuration where two GPIOs are directly wired together.

6.2.2 Supported platforms

Table 47. Supported platforms for gpio_basic_api

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53
i.MX 93	i.MX 93 EVK	Cortex-A55

6.2.3 Running the tests

This section describes the steps for running the gpio_basic_api test on the supported boards (i.MX 93 EVK, i.MX 8M Plus EVK, i.MX 8M Mini EVK, and i.MX 8M Nano EVK).

6.2.3.1 Running the test on i.MX 93 EVK board

- **Hardware setup**

On the i.MX 93 EVK board, connect the port0 to port27 of the GPIO2 controller IOs to the expansion connector J1001 EXP_GPIO_IO0 to EXP_GPIO_IO27.

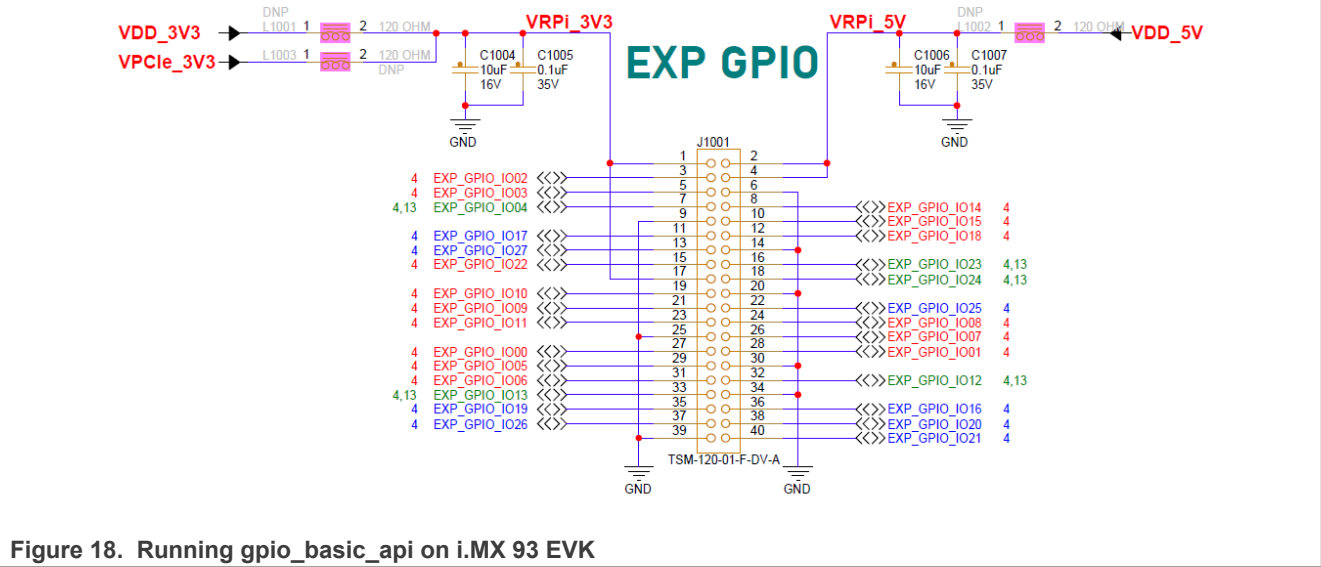

Figure 18. Running gpio_basic_api on i.MX 93 EVK

The IO port13 and port14 of the GPIO2 controller are used for testing. Therefore, you must connect the Expansion Connector's Pin 8 (EXP_GPIO_IO14) together with Pin 33 (EXP_GPIO_IO13) with a Dupont Line.

- **Software setup**

The following DTS overlay is used to specify the GPIO IO ports to be used:

```
tests/drivers/gpio/gpio_basic_api/boards/imx93_evk_mimx9352_a55.overlay
```

- **Building the application**

```
west build -p always -b imx93_evk/mimx9352/a55 tests/drivers/gpio/gpio_basic_api
```

- **Running the Test**
  - Refer to the section Section 6.2.3.5 to view the testing log.

  .

6.2.3.2 Running the test on i.MX 8M Plus EVK board

- **Hardware setup**

On i.MX 8M Plus EVK, PCA6416 is used for IO expansion by using I2C3 as control bus and GPIO1 controller's IO port12 as interrupt line. Expansion IO ports of PCA6416 are connected to the J21 expansion connector.

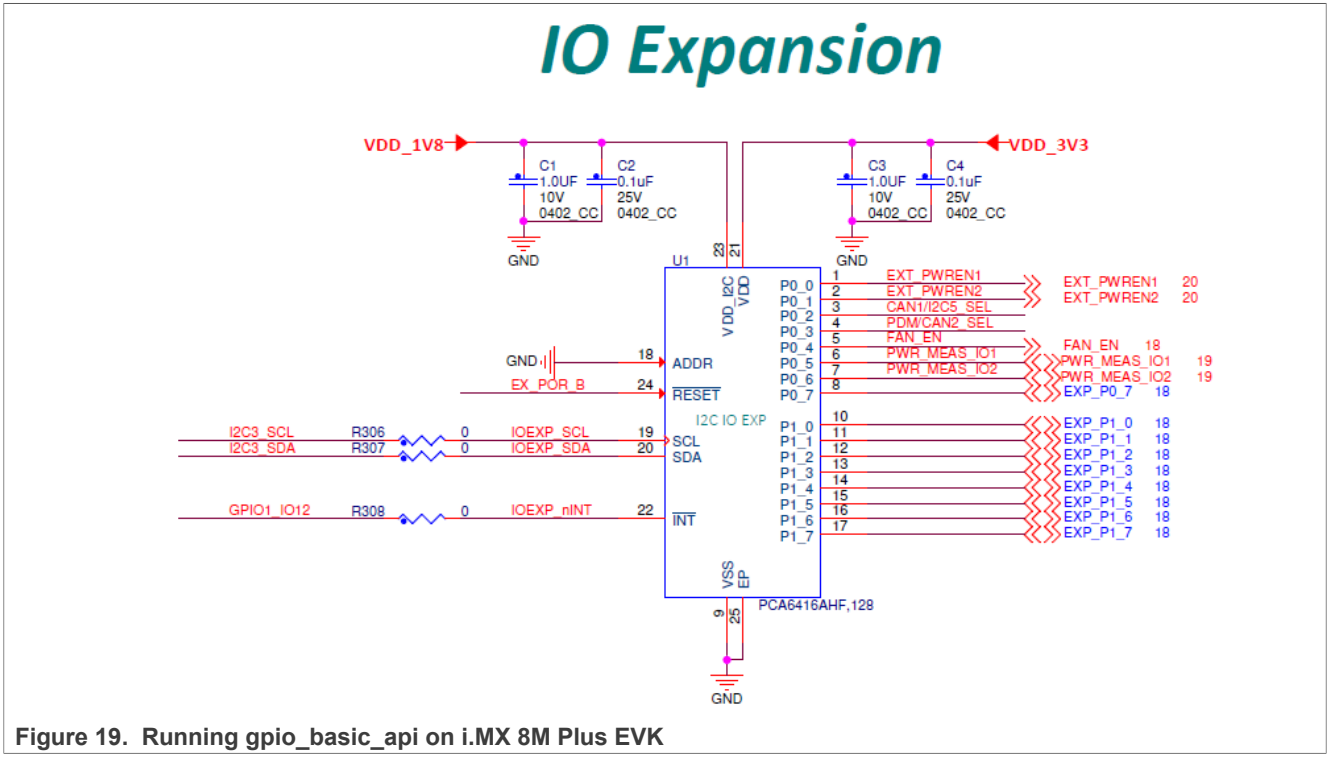

Figure 19. Running gpio_basic_api on i.MX 8M Plus EVK

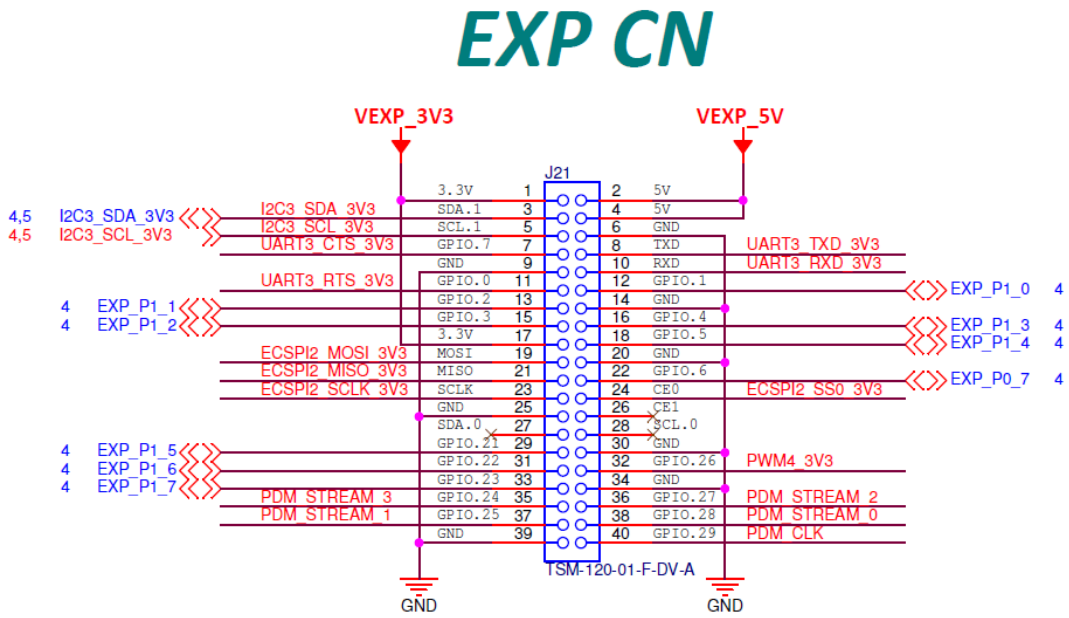

Figure 20. Expansion connector on i.MX 8M Plus EVK board

Use the Pin12 (EXP_P1_0) and Pin16 (EXP_P1_3) of J21 connector to two GPIO lines to run the `gpio_basic_api` testing. This process can cover the testing both for drivers of I2C, GPIO, and PCA6416. For this you must connect the expansion connector's Pin12 (EXP_P1_0) and Pin16 (EXP_P1_3) together with a Dupont Line.

- **Software setup**: The following DTS overlay must used to specify the GPIO IO ports to be used:

```
tests/drivers/gpio/gpio_basic_api/boards/imx8mp_evk_mimx8ml8_a53.overlay
```

- **Building the application:**

```
west build -p always -b imx8mp_evk/mimx8ml8/a53tests/drivers/gpio/
gpio_basic_api
```

- **Running the Test**: Refer to the section Section 6.2.3.5 to view the testing log.

6.2.3.3 Running the test on i.MX 8M Mini EVK

- **Hardware Setup**

On i.MX 8M Mini EVK board, PCA6416 is used for IO expansion by using I2C3 as control bus and GPIO1 controller's IO port12 as interrupt line IOEXP_nINT. Some of the expansion IO ports of PCA6416 are connected to the J1003 expansion connector.

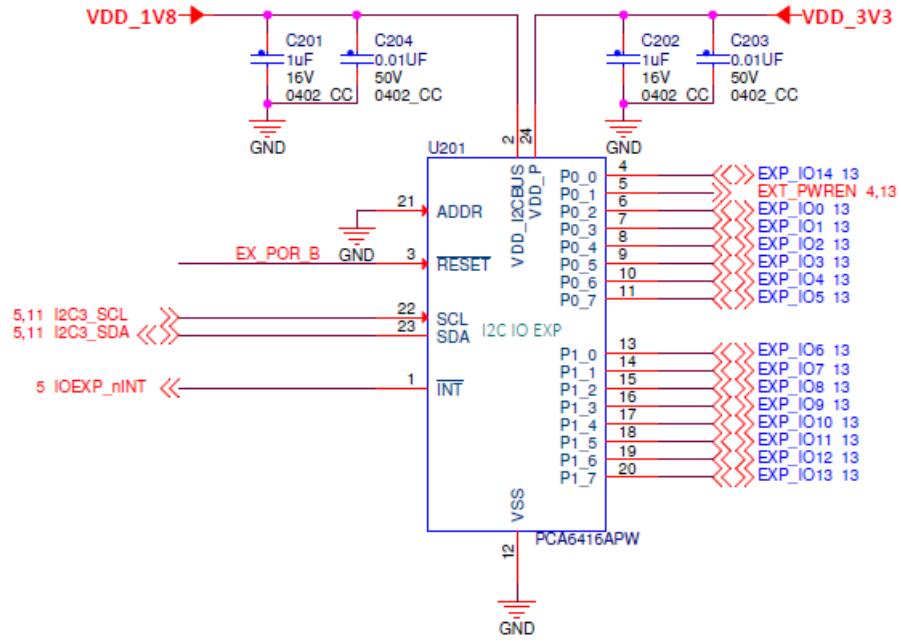

Figure 21. Running gpio_basic_api test on i.MX 8M Mini EVK board

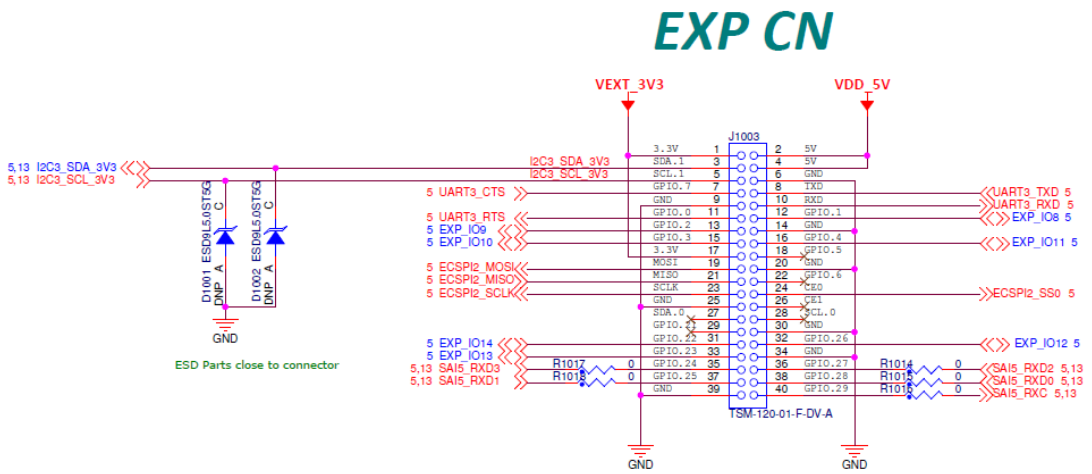

Figure 22. Expansion connector J1003 on i.MX 8M Mini EVK board

Use J1003 connector's Pin12 (EXP_IO8) and Pin16 (EXP_IO11) as two GPIO lines to run the gpio_basic_api testing. This step can cover the testing both for drivers of I2C, GPIO, and PCA6416. For this, connect J1003 expansion connector's Pin12 (EXP_IO8) and Pin16 (EXP_IO11) together with a Dupont Line.

- **Software Setup**: Use the following DTS overlay to specify the GPIO IO ports to be used:

```
tests/drivers/gpio/gpio_basic_api/boards/imx8mm_evk_mimx8mm6_a53.overlay
```

- **Building the application**

```
west build -p always -b imx8mm_evk/mimx8mm6/a53 tests/drivers/gpio/
gpio_basic_api
```

- **Running the Test**: Refer to the section Section 6.2.3.5 to view the testing log.

6.2.3.4 Run on i.MX 8M Nano EVK

- **Hardware Setup**

On i.MX 8M Nano EVK, PCA6416 is used for IO expansion by using I2C3 as control bus and GPIO1 controller's IO port12 as interrupt line IOEXP_nINT. Some of Expansion IO ports of PCA6416 are connected to J1003 expansion connector.

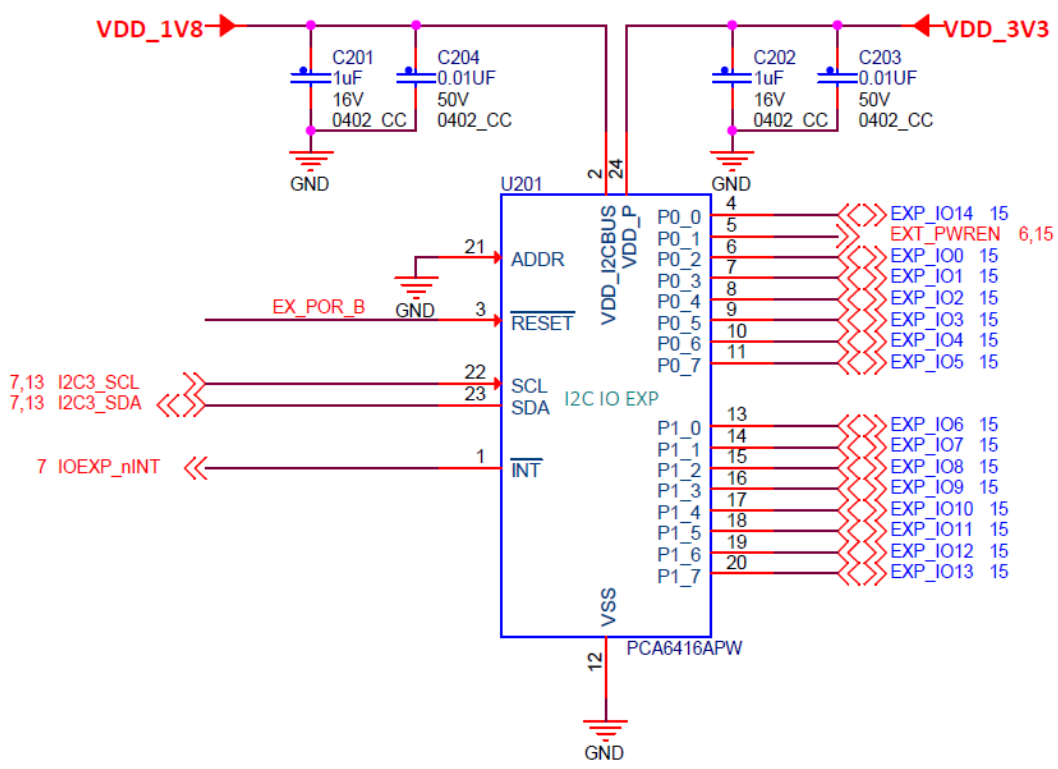

Figure 23. Running gpio_basic_api on i.MX 8M Nano EVK

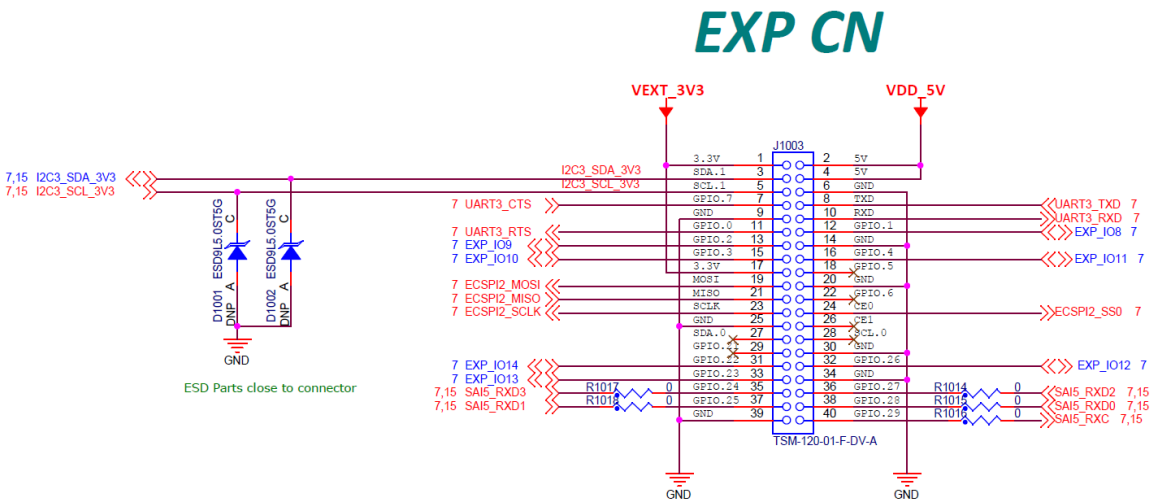

Figure 24. Expansion connector J1003 on i.MX 8M Nano EVK board

Use J1003 connector's Pin12 (EXP_IO8) and Pin16 (EXP_IO11) as two GPIO lines to run the gpio_basic_api testing, it can cover the testing both for drivers of I2C, GPIO and PCA6416. So need to connect J1003 expand connector's Pin12 (EXP_IO8) and Pin16 (EXP_IO11) together with a Dupont Line.

- **Software Setup**:
  Use the following DTS overlay to specify the GPIO IO ports to be used:

```
tests/drivers/gpio/gpio_basic_api/boards/imx8mn_evk_mimx8mn6_a53.overlay
```

- **Building the application**

```
west build -p always -b imx8mn_evk/mimx8mn6/a53 tests/drivers/gpio/
gpio_basic_api
```

- **Running the Test**: Refer to the section Section 6.2.3.5 to view the testing log.

6.2.3.5 Test logs

Refer to the following log for testing. The log might change on different platforms and with a different Zephyr version.

```
*** Booting Zephyr OS build v3.6.0-963-g06dc02cc432a ***
Running TESTSUITE after_flash_gpio_config_trigger
===================================================================
START - test_gpio_config_trigger
PASS - test_gpio_config_trigger in 0.011 seconds
===================================================================
START - test_gpio_config_twice_trigger
PASS - test_gpio_config_twice_trigger in 0.011 seconds
===================================================================
TESTSUITE after_flash_gpio_config_trigger succeeded
Running TESTSUITE gpio_port
===================================================================
START - test_gpio_port
Validate device gpio@43810000
Check gpio@43810000 output 13 connected to input 14
OUT 13 to IN 14 linkage works
- bits_physical
- pin_physical
- check_raw_output_levels
- check_logic_output_levels
- check_input_levels
- bits_logical
- check_pulls
PASS - test_gpio_port in 0.023 seconds
```

```
===================================================================
TESTSUITE gpio_port succeeded
Running TESTSUITE gpio_port_cb_mgmt
===================================================================
START - test_gpio_callback_add_remove
callback_2 triggered: 1
callback_1 triggered: 1
callback_2 triggered: 1
PASS - test_gpio_callback_add_remove in 3.601 seconds
===================================================================
START - test_gpio_callback_enable_disable
callback_2 triggered: 1
callback_1 triggered: 1
```

```
callback_2 triggered: 1
callback_1 triggered: 1
PASS - test_gpio_callback_enable_disable in 3.601 seconds
===================================================================
START - test_gpio_callback_self_remove
callback_remove_self triggered: 1
callback_1 triggered: 1
callback_1 triggered: 1
PASS - test_gpio_callback_self_remove in 2.501 seconds
===================================================================
TESTSUITE gpio_port_cb_mgmt succeeded
Running TESTSUITE gpio_port_cb_vari
===================================================================
START - test_gpio_callback_variants
callback triggered: 1
OUT init a0001, IN cfg 3400000, cnt 1
callback triggered: 1
OUT init 60000, IN cfg 5400000, cnt 1
callback triggered: 1
OUT init 60000, IN cfg 5c00000, cnt 1
callback triggered: 1
OUT init a0001, IN cfg 3c00000, cnt 1
callback triggered: 1
callback triggered: 2
callback triggered: 3
OUT init 60000, IN cfg 4400000, cnt 3
callback triggered: 1
callback triggered: 2
callback triggered: 3
OUT init a0001, IN cfg 2400000, cnt 3
callback triggered: 1
callback triggered: 2
callback triggered: 3
OUT init 60000, IN cfg 4c00000, cnt 3
callback triggered: 1
callback triggered: 2
callback triggered: 3
OUT init a0001, IN cfg 2c00000, cnt 3
callback triggered: 1
callback triggered: 2
OUT init a0001, IN cfg 7400000, cnt 2
PASS - test_gpio_callback_variants in 9.931 seconds
===================================================================
TESTSUITE gpio_port_cb_vari succeeded
```

```
------ TESTSUITE SUMMARY START ------
SUITE PASS - 100.00% [after_flash_gpio_config_trigger]: pass = 2, fail = 0, skip
 = 0, total = 2 duration = 0.022 seconds
    - PASS - [after_flash_gpio_config_trigger.test_gpio_config_trigger] duration
 = 0.011 seconds
    - PASS - [after_flash_gpio_config_trigger.test_gpio_config_twice_trigger]
 duration = 0.011 seconds
SUITE PASS - 100.00% [gpio_port]: pass = 1, fail = 0, skip = 0, total = 1
 duration = 0.023 seconds
    - PASS - [gpio_port.test_gpio_port] duration = 0.023 seconds
SUITE PASS - 100.00% [gpio_port_cb_mgmt]: pass = 3, fail = 0, skip = 0, total =
 3 duration = 9.703 seconds
    - PASS - [gpio_port_cb_mgmt.test_gpio_callback_add_remove] duration = 3.601
 seconds
```

```
    - PASS - [gpio_port_cb_mgmt.test_gpio_callback_enable_disable] duration =
 3.601 seconds
    - PASS - [gpio_port_cb_mgmt.test_gpio_callback_self_remove] duration = 2.501
 seconds
SUITE PASS - 100.00% [gpio_port_cb_vari]: pass = 1, fail = 0, skip = 0, total =
 1 duration= 9.931 seconds
    - PASS - [gpio_port_cb_vari.test_gpio_callback_variants] duration = 9.931
 seconds
------ TESTSUITE SUMMARY END ------
===================================================================
PROJECT EXECUTION SUCCESSFUL
```

6.3 button and blinky

6.3.1 Overview

The zephyr kernel has some built-in sample applications that can be used for GPIO testing.

“`samples/basic/button`” is a simple button demo showcasing the use of GPIO input with interrupts. For this purpose, a button with alias “sw0” needs to be configured in dts. An optional “led0” devicetree alias can also be used. When the `led0` button is pressed, it displays a message on the log console and LED turns on.

The code “`samples/basic/blinky`” blinks an LED specified by the “`led0`” dts node alias forever by toggling the GPIO pin connected to this LED.

6.3.2 Supported platforms

Table 48. Supported platforms for button and blinky

Platform	Boards	CPU Core
i.MX 93	i.MX 93 EVK	Cortex-A55

6.3.3 Running the tests

This section describes the steps for running the button and blinky test on the supported board (i.MX 93 EVK).

6.3.3.1 Running button and blinky on i.MX 93 EVK

On the i.MX 93 EVK board, there are two buttons: SW1003 and SW1004. SW1003 is connected to RFU_BTN1 and SW1004 is connected to RFU_BTN2. These buttons are meant for customized use cases.

By default, RFU_BTN1 and RFU_BTN2 signals are connected to the GPIO2 IO port23 (EXP_GPIO_IO23) and port24 (EXP_GPIO_IO24). By configuring switch SW1005 to value ‘1010’, these can be connected to EXP_BTN1 and EXP_BTN2, which are connected to the onboard GPIO expander, PCAL6524.

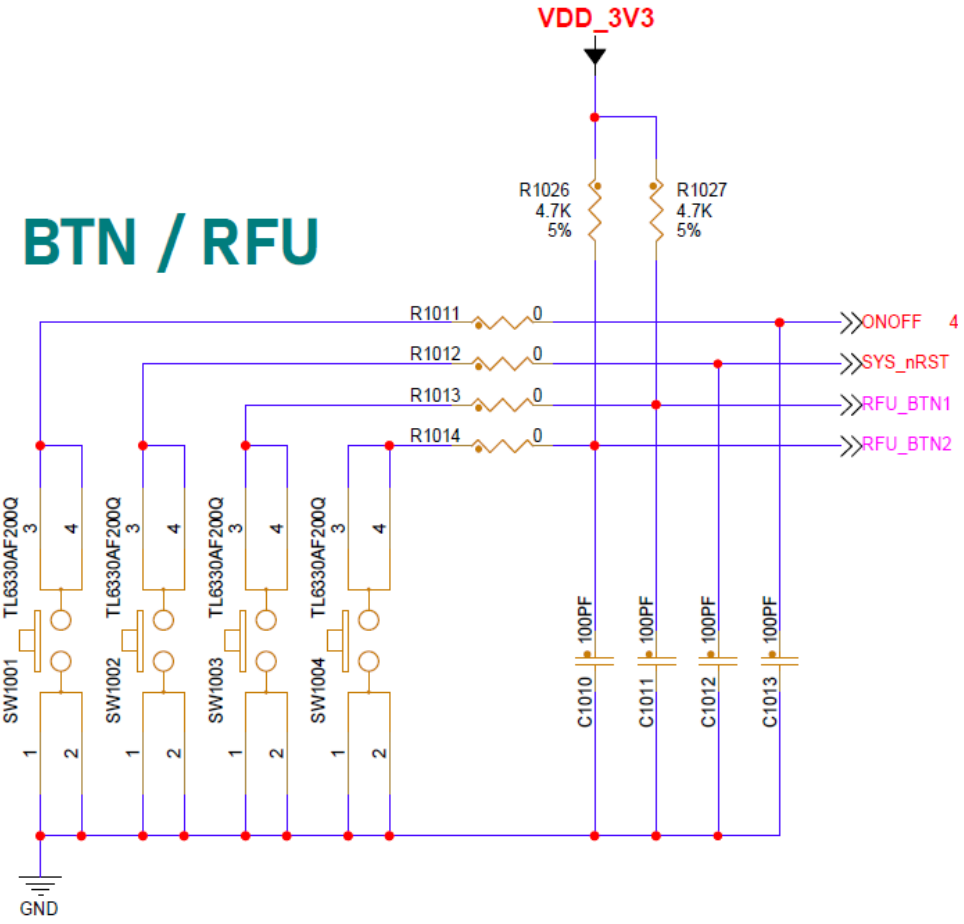

Figure 25. SW1001 to SW1004 on the i.MX 93 EVK board

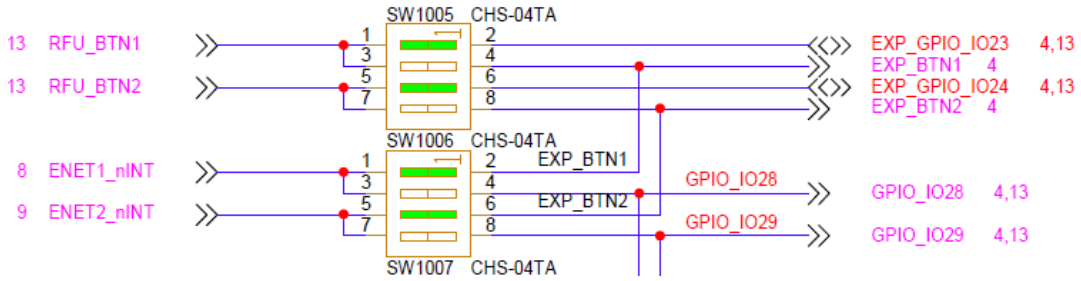

Figure 26. SW1005, SW1006, and SW1007 on the i.MX 93 EVK board

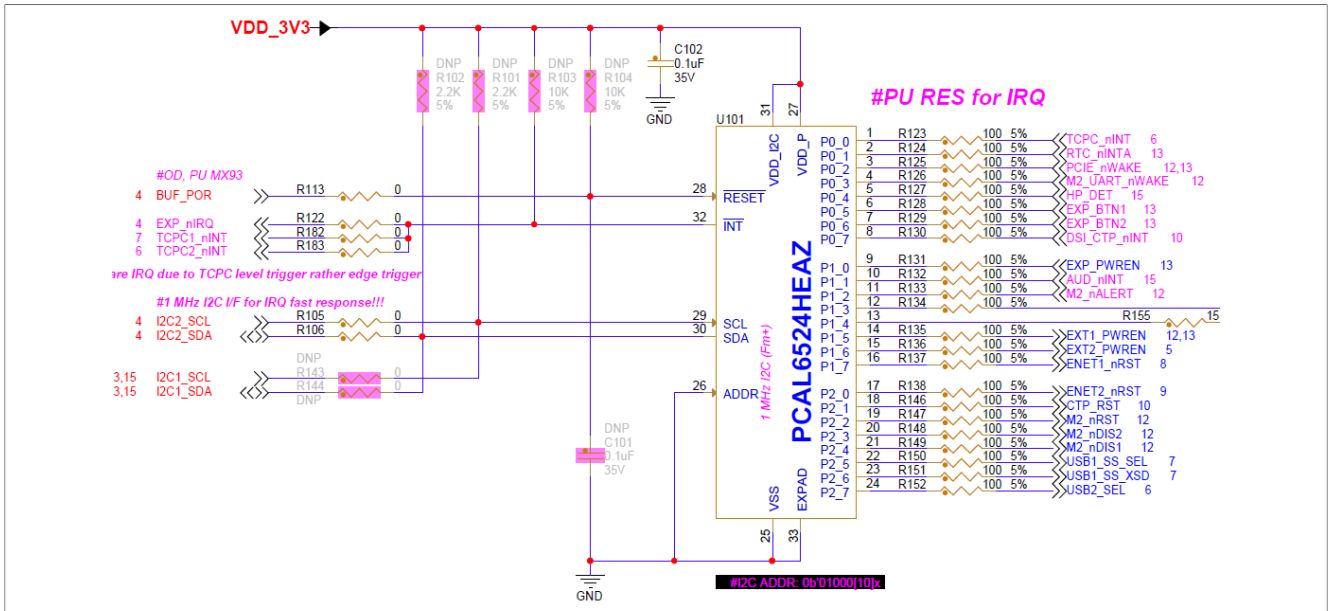

Figure 27. GPIO expander PCAL6524 on the i.MX 93 EVK board

LED D1001 on the i.MX 93 EVK can be controlled by the GPIO2 IO ports: port4, port12, and port13.

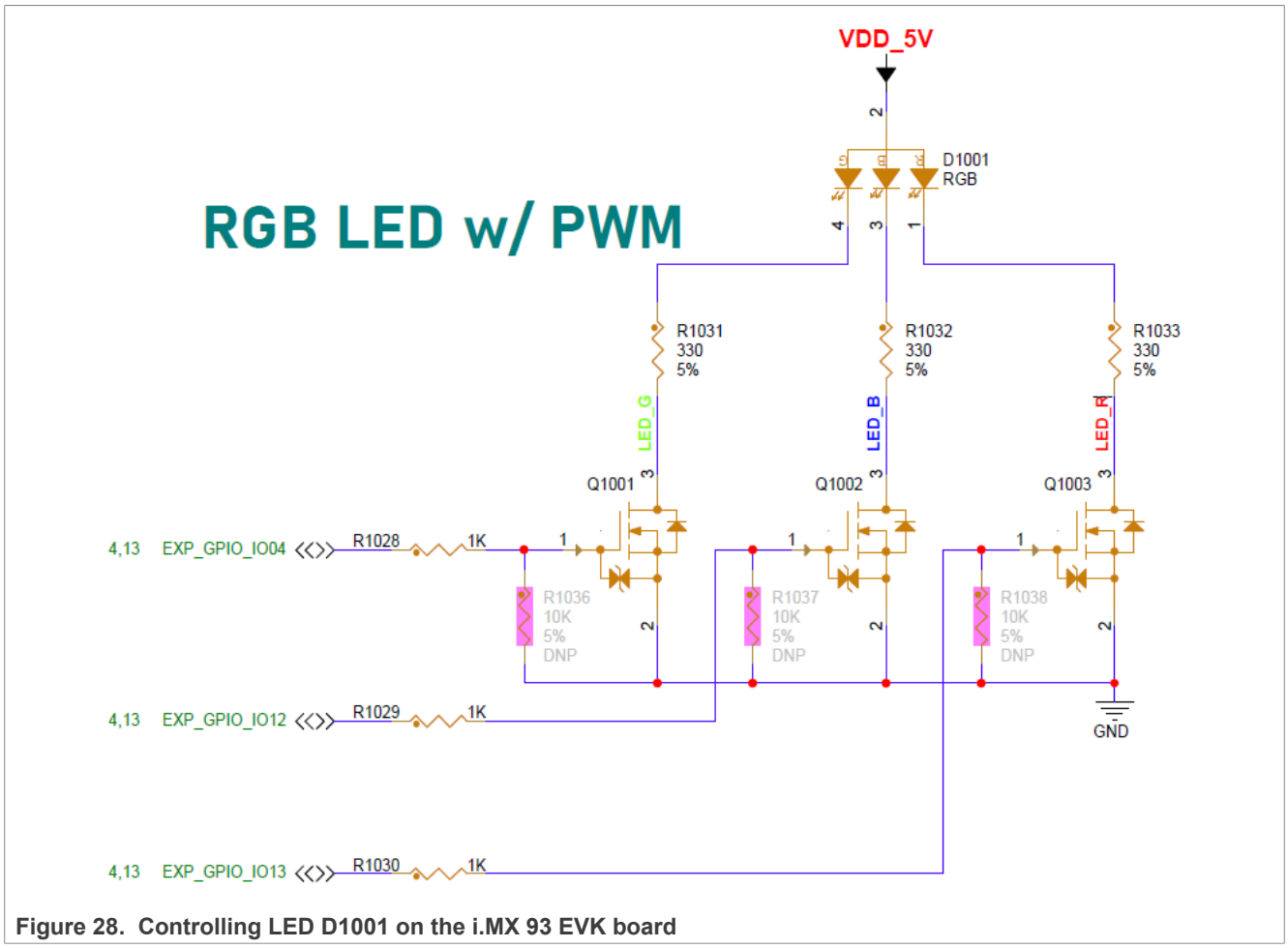

Figure 28. Controlling LED D1001 on the i.MX 93 EVK board

- **Setup for using on-chip GPIO signals**
  1. **Hardware setup**: Set the switch SW1005 to value ‘1010’.
  2. **Software setup** Button and LED device nodes are added in the board device tree:

     ```
     boards/arm64/mimx93_evk/mimx93_evk_a55.dts.
     ```

  3. **Building the Testing Applications**

     ```
     west build -p always -b imx93_evk/mimx9352/a55 samples/basic/blinky
     west build -p always -b imx93_evk/mimx9352/a55 samples/basic/button
     ```

  4. **Setup for using Expander GPIO signals**
  5. **Hardware setup**: Set the switch SW1006 to 0000, then switch SW1005 to 0101.
  6. **Software setup**: Dts overlay is provided in `boards/nxp/imx93_evk/dts/imx93_evk_mimx9352_exp_btn.overlay`

1. **Building the testing applications**

   ```
   west build -p always -b imx93_evk/mimx9352/a55 samples/basic/blinky
   west build -p always -b imx93_evk/mimx9352/a55 samples/basic/button -
   DEXTRA_DTC_OVERLAY_FILE=imx93_evk_mimx9352_exp_btn.overlay
   ```

- **Running the applications**
  Execute the following command in i.MX 93 EVK boot into the U-Boot command line to start the Zephyr application.

  ```
  tftp 0xD0000000 izephyr.bin;
  dcache flush; icache flush;
  cpu 1 release 0xD0000000
  ```

Note: *In Zephyr kernel v2.5 or older version, Zephyr kernel uses the DDR memory start from 0xC0000000, in such case, replace “0xD0000000” with “0xC0000000” in this command. Check the Zephyr board documentation for confirmation.*

1. Running the “`samples/basic/blinky`” application
   When the application starts, the LED D1001 blinks forever with the following console log:

   ```
   *** Booting Zephyr OS build v3.6.0-961-gddb147d0a45d ***
   LED state: OFF
   LED state: ON
   LED state: OFF
   LED state: ON
   LED state: OFF
   LED state: ON
   LED state: OFF
   ```

2. Running the “`samples/basic/button`” application.
   When the application starts, pressing the button SW1003 turns on the LED D1001. The LED D1001 turns off when the button is released. The console log is as follows; it displays the hardware clock cycles when the button is pressed.

   ```
   *** Booting Zephyr OS build v3.6.0-961-gddb147d0a45d ***
   Set up button at gpio@43810000 pin 23
   Set up LED at gpio@43810000 pin 13
   Press the button
   Button pressed at 444008942
   Button pressed at 478978442
   Button pressed at 505319416
   Button pressed at 527666298
   ```

```
Button pressed at 549892679
Button pressed at 570650726
```

6.4 Multithread Blinky

6.4.1 Overview

The use case "`samples/basic/threads`" demonstrates spawning multiple threads using `K_THREAD_DEFINE()`. It spawns three threads. Each thread is then defined at compile time using `K_THREAD_DEFINE`.

The first two each control an LED. These LEDs, led0 and led1, have loop control and timing logic controlled by separate functions.

```
blink0() controls led0 and has a 100ms sleep cycle
blink1() controls led1 and has a 1000ms sleep cycle
```

When either of these threads toggles its LED, it also pushes information into a FIFO identifying the thread/LED and how many times it has been toggled.

The third thread uses the `printk()` command to print the information added to the FIFO to the device console.

Refer to https://docs.zephyrproject.org/latest/samples/basic/threads/README.html for details.

6.4.2 Supported platforms

Table 49. Supported platforms for multi-thread blinky

Platform	Boards	CPU Core
i.MX 93	i.MX 93 EVK	Cortex-A55

6.4.3 Running the tests

This section describes the steps for running the `multithread blinky` test on the supported board (i.MX 93 EVK).

6.4.3.1 Running the test on i.MX 93 EVK

- **Hardware Setup**
  This demo uses the red LED and green LED of the RGB LED D1001. The LED D1001 on the i.MX 93 EVK can be controlled by three GPIO2 IO ports:- port4, port12, and port13.

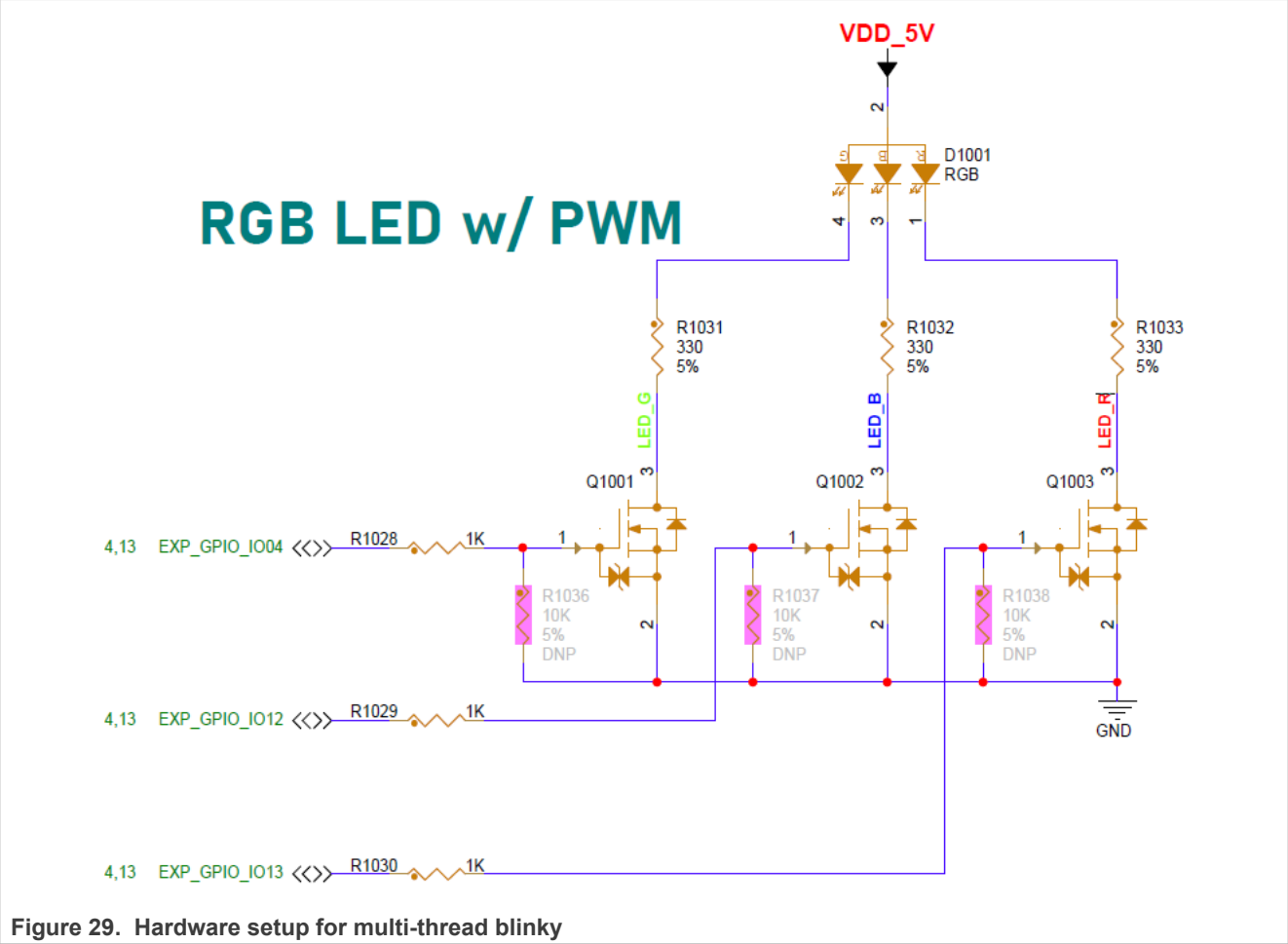

Figure 29. Hardware setup for multi-thread blinky

- **Software setup**
  The demo uses two LED “led0” and “led1” specified by device tree alias, they have been added in the board device tree:

```
boards/arm64/mimx93_evk/mimx93_evk_a55.dts
```

- **Building the application**

```
west build -p always -b imx93_evk/mimx9352/a55 samples/basic/threads
```

- **Running the application**
  When the application starts, D1001 LED blinks with Red color and Green color, and the console log as follows is displayed. As '`led1`' sleep cycle time is ten times of `led0`. Therefore, we can infer from the following log that the led0 blink times is ten times of led1.

```
*** Booting Zephyr OS build v3.6.0-961-gddb147d0a45d ***
Toggled led0; counter=0
Toggled led1; counter=0
Toggled led0; counter=1
Toggled led0; counter=2
Toggled led0; counter=3
Toggled led0; counter=4
Toggled led0; counter=5
Toggled led0; counter=6
Toggled led0; counter=7
```

```
Toggled led0; counter=8
Toggled led0; counter=9
Toggled led1; counter=1
Toggled led0; counter=10
Toggled led0; counter=11
Toggled led0; counter=12
Toggled led0; counter=13
Toggled led0; counter=14
Toggled led0; counter=15
Toggled led0; counter=16
Toggled led0; counter=17
Toggled led0; counter=18
Toggled led0; counter=19
Toggled led1; counter=2
Toggled led0; counter=20
Toggled led0; counter=21
Toggled led0; counter=22
Toggled led0; counter=23
```

6.5 zperf

6.5.1 Overview

zperf is a shell utility which allows to generate network traffic in Zephyr. The tool may be used to evaluate network bandwidth. zperf is compatible with iPerf_2.0.5.

For more details, refer to:

https://docs.zephyrproject.org/latest/connectivity/networking/api/zperf.html

6.5.2 Supported platforms

Table 50.

Platform	Boards	CPU Core
i.MX 8M Mini	i.MX 8M Mini EVK	Cortex-A53
i.MX 8M Plus	i.MX 8M Plus EVK	Cortex-A53
i.MX 8M Nano	i.MX 8M Nano EVK	Cortex-A53
i.MX 93	i.MX 93 EVK	Cortex-A55

6.5.3 Hardware setup

Connect the Ethernet port on the board to another PC or networking router.

6.5.4 Software setup and building the application

Building zperf application:

- For i.MX 8M Plus EVK:

```
west build -p always -b imx8mp_evk/mimx8ml8/a53 samples/net/zperf/
```

- For i.MX 8M Mini EVK:

```
west build -p always -b imx8mm_evk/mimx8mm6/a53 samples/net/zperf/
```

- For i.MX 8M Nano EVK:

```
west build -p always -b imx8mn_evk/mimx8mn6/a53 samples/net/zperf/
```

- For i.MX 93 EVK

```
west build -p always -b imx93_evk/mimx9352/a55 samples/net/zperf/
```

6.5.5 Running the test

Take i.MX 8M Plus EVK as example, after Zephyr bootup, it displays the following log:

```
*** Booting Zephyr OS build v4.0.0-2779-g1be0c78915cf ***
[00:00:00.016,000] <inf> net_config: Initializing network
[00:00:00.016,000] <inf> net_config: Waiting interface 1 (0xc005d5a0) to be
 up...
[00:00:03.516,000] <inf> phy_rt_rtl8211f: PHY 1 is up
[00:00:03.516,000] <inf> phy_rt_rtl8211f: PHY (1) Link speed 1000 Mb, full
 duplex
[00:00:03.516,000] <inf> eth_nxp_enet_mac: Link is up
[00:00:03.516,000] <inf> net_config: Interface 1 (0xc005d5a0) coming up
[00:00:03.516,000] <inf> net_config: IPv4 address: 192.0.2.1
[00:00:03.617,000] <inf> net_config: IPv6 address: 2001:db8::1
[00:00:03.617,000] <inf> net_config: IPv6 address: 2001:db8::1
uart:~$
```

Then configure the IP address and use ping to test whether the networking works:

```
uart:~$ net ipv4
IPv4 support                              : enabled
IPv4 fragmentation support                : disabled
IPv4 conflict detection support           : disabled
Path MTU Discovery (PMTU)                 : disabled
Max number of IPv4 network interfaces in the system          : 1
Max number of unicast IPv4 addresses per network interface   : 1
```

```
Max number of multicast IPv4 addresses per network interface : 2
```

```
IPv4 addresses for interface 1 (0xc005d5a0) (Ethernet)
======================================================
Type            State           Ref     Address
manual          preferred       1       192.0.2.1/255.255.255.0
uart:~$ net ipv4 del 1 192.0.2.1
uart:~$ net ipv4 add 1 10.193.20.39 255.255.0.0
uart:~$ net ipv4
IPv4 support                              : enabled
IPv4 fragmentation support                : disabled
IPv4 conflict detection support           : disabled
Path MTU Discovery (PMTU)                 : disabled
Max number of IPv4 network interfaces in the system          : 1
Max number of unicast IPv4 addresses per network interface   : 1
Max number of multicast IPv4 addresses per network interface : 2
```

```
IPv4 addresses for interface 1 (0xc005d5a0) (Ethernet)
======================================================
Type            State           Ref     Address
manual          preferred       1       10.193.20.39/255.255.0.0
uart:~$ net ping 10.193.20.18
PING 10.193.20.18
28 bytes from 10.193.20.18 to 10.193.20.39: icmp_seq=1 ttl=64 time=0.37 ms
28 bytes from 10.193.20.18 to 10.193.20.39: icmp_seq=2 ttl=64 time=0.19 ms
28 bytes from 10.193.20.18 to 10.193.20.39: icmp_seq=3 ttl=64 time=0.26 ms
```

7 Note about the source code in the document

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

Copyright 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 must be 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.

8 Revision history

Table 51 summarizes the revisions to this document.

Table 51. Document revision history

Document ID	Release date	Description
UG10199 v.1.0	26 March 2025	Initial public release

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.

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.

Contents