FreeRTOS on FSiMX8MM Boards

Manual on how to use/configuring the software

Version 1.2

26.02.2026

Elektronik Systeme

© F&S Elektronik Systeme GmbH
Untere Waldplätze 23
D-70569 Stuttgart
Germany

Phone: +49(0)711-123722-0
Fax: +49(0)711-123722-99

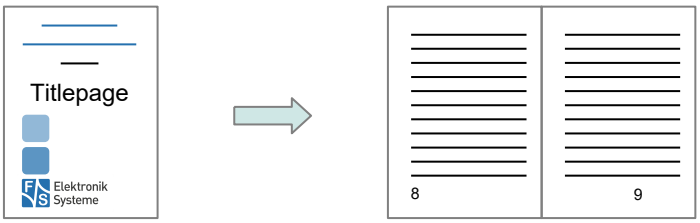
About This Document

This document describes how to configure the Linux kernel, the device tree and the board to use it with FreeRTOS and its demo applications provided. The software is configured for PicoCoreMX8MM from F&S under Linux/Buildroot.

Remark

The version number on the title page of this document is the version of the document. It is not related to the version number of any software release. The latest version of this document can always be found at http://www.fs-net.de.

How to Print This Document



This document is designed to be printed double-sided (front and back) on A4 paper. If you want to read it with a PDF reader program, you should use a two-page layout where the title page is an extra single page. The settings are correct if the page numbers are at the outside of the pages, even pages on the left and odd pages on the right side. If it is reversed, then the title page is handled wrongly and is part of the first double-page instead of a single page.

Typographical Conventions

We use different fonts and highlighting to emphasize the context of special terms:

`File names`

Menu entries

`Board input/output`

`Program code`

`PC input/output`

`Listings`

`Generic input/output`

`Variables`

Hints and information

History

Date	V	Platform	A,M,R	Chapter	Description	Au
2020	1.0	All	A	-	Derivate from MX7ULP-Doku	DD
2021	1.1	All	A, M	-	Update SDK to version 2.9.1	AD
				4.5.1	Add Chapter “Additional binaries”	AD
				8.1	Add entry regarding SDMA usage	AD
				8.4.12	Add information regarding NBoot and patches	AD
				8.4.3	Move gpt_capture example to not_tested	AD
				8.4.6	Add pwm, freertos_i2c examples and hardware_flow_contol	AD
				8.6.5		AD
				9	Update figures and tables	AD
2026	1.2	All	A, M, R	-	Update SDK to version 25.3	RM
					Remove Configuration for Cortex-M4 usage Chapter	RM
				1.2	Add pin Assignment for FS 8MM OSM-SF	RM
				5.1	Add section build.sh to build the examples	RM
				5.2	Add section clean_all.sh to clean the examples	RM
				6	Modify adding costume Board Chapter	RM
				7.1	Add section Execute in U-Boot	RM
				7.2	Add section Execute in Linux	RM
				7.3	Add section Execute using a SEGGER J-Link Debugger	RM
				8	Add freertos_sem_static. Remove freertos_i2c and freertos_uart	RM
				8.1	Remove device tree note for the SDMA examples	TK
				-	Add U-Boot environment variable for mcore clock	TK

V	Version
A,M,R	Added, Modified, Removed
Au	Author

Table of Contents

1 Pin Assignment

In the following subchapters you can find an overview which pins are used for each Board. The examples itself also contain the necessary pins.

1.1 PicoCoreMX8MM / MX

1.1.1 GPIOS

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
SPI_B_MOSI / LED	J1_60		J11_6

1.1.2 ECSPI

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1_58		J11_5
SPI_B_MOSI	J1_60		J11_6

SPI_B_SCLK	J1_62	J11_3
SPI_B_SS0	J1_64	J11_4
GND	---	J11_11

1.1.3 I2C

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
I2C_A_SCL	J1_4		J11_16
I2C_A_SDA	J1_6		J11_17
GND	---		J11_11

1.1.4 PWM

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
PWM	J2_63		J11_34

1.1.5 UART_B

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
UART_B_TXD	J1_28		J10_5
UART_B_RXD	J1_26		J10_3
UART_B_CTS	J1_24		J10_6
UART_B_RTS	J1_22		J10_4

1.2 FS 8MM OSM-SF

1.2.1 GPIOS

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.4
SPI_B_MOSI / LED	J1I_Y23	J1_60	J11_6

1.2.2 ECSPI

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1I_Y22	J1_58	J11_5
SPI_B_MOSI	J1I_Y23	J1_60	J11_6
SPI_B_SCLK	J1I_Y21	J1_62	J11_3
SPI_B_SS0	J1I_AA23	J1_64	J11_4
GND	---		J11_11

1.2.3 I2C

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.3
I2C_A_SCL	J1I_AA15	J1_4	J11_16
I2C_A_SDA	J1I_AA15	J1_6	J11_17
GND	---		J11_11

1.2.4 PWM

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.3
PWM	J3J_F18	J2_63	J11_34

2 Introduction

The F&S FreeRTOS_BSP-package is based on the MCUXpresso Software Development Kit (SDK) by NXP. It provides comprehensive software support for Kinetis and LPC Microcontrollers.

The MCUXpresso SDK includes a flexible set of peripheral drivers designed to speed up and simplify development of embedded applications. Along with the peripheral drivers, the MCUXpresso SDK provides an extensive and rich set of example applications covering everything from basic peripheral use case examples to full demo applications. The MCUXpresso SDK contains FreeRTOS and various other middleware to support rapid development.

3 Installation

This section describes the installation of the CST code-signing client files.

3.1 Installation of the GCC embedded toolchain

The examples are tested and can be built with the GCC embedded toolchain (gcc-arm-none-eabi-10.3-2021-10-update-x86_64-linux), which can be found under developer.arm.com.

If the toolchain is not installed, you have to download the file and extract the content to your filesystem:

```
tar -xvjf gcc-arm-none-eabi-${version}.tar.bz2
```

where ${version} will be replaced by the corresponding version you've downloaded.

It is necessary to export the `ARMGCC_DIR` environment variable, if it´s not already exported:

```
export ARMGCC_DIR=/usr/local/arm/gcc-arm-none-eabi-${version}
```

For a more convenient way you can add this to the rc file of your favorite shell (e.g. zshrc, bashrc, etc.)

3.2 Download Source Code

To download FreeRTOS source code, go to the F&S main website

http://www.fs-net.de

First you have to register with the website. Click on *Login* right at the top of the window and on the text “I am not registered, yet. Register now” (*Figure 1*).

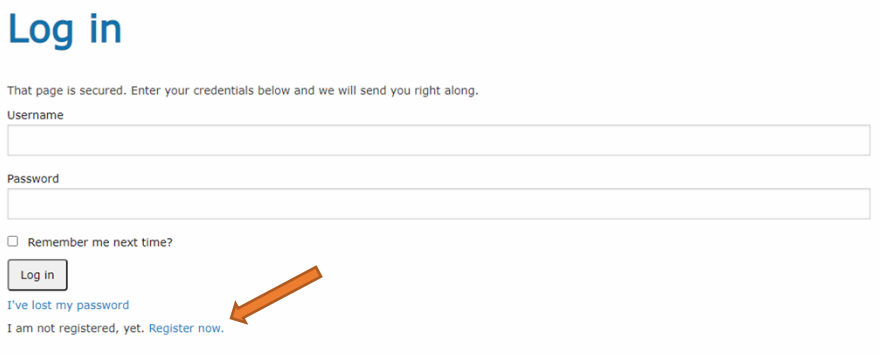


Figure 1: Register with F&S website

In the screen appearing now, fill in all fields and then click on *Register*. You are now registered and can use the personal features of the website, for example the Support Forum and downloading software.

After logging in, you are at your personal page, called “My F&S”. You can always reach this place by selecting *Support → My F&S* from the top menu. Here you can find all software downloads that are available for you. In the top sections there are private downloads for you or your company (may be empty) and in the bottom section you will find generic downloads for all registered customers.

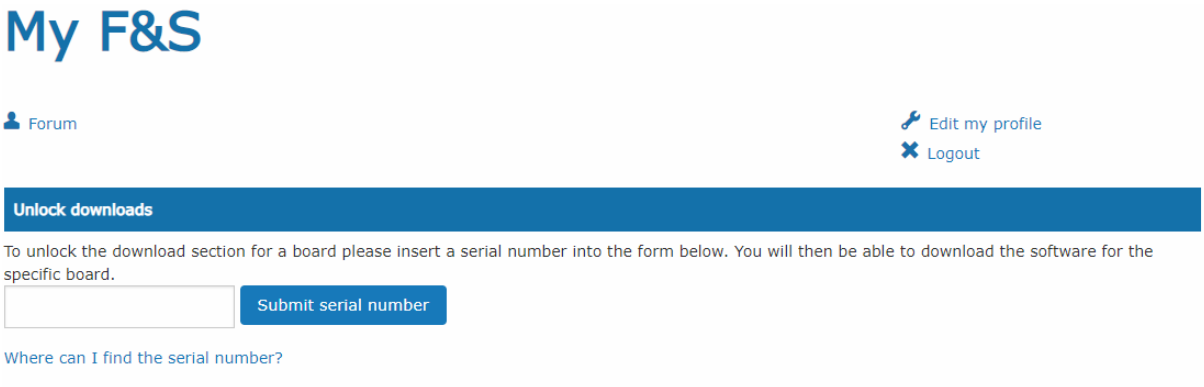


Figure 2: Unlock software with serial number

To get access to the software of a specific board, you have to enter the serial number of one of these boards (see *Figure 2*). Click on “Where can I find the serial number” to get pictures of examples where to find this number on your product. Enter the number in the white field and press *Submit serial number*. This enables the software section for this board type for you. You will find Linux, Windows CE, and all other software and tools available for this platform like `DCUTerm` or `NetDCUUsbLoader`.

First click on the type of your board, e.g. PicoCoreMX8MM, then on Linux. Now click on FreeRTOS. This will bring up a list of all our FreeRTOS releases. Old releases up to 2019 had <x>.<y> as version identifier, new releases use V<year>.<month>. We will abbreviate this as <v> from now on. Select the newest version, for example *freertos-sdk-2.9.1-fsimx8mm-V2021.07*. This will finally show two archives that can be downloaded.

When you look at our Linux releases, you will find a list of all our releases and a README text. There are usually two files related to a release.

freertos-sdk-25.03.00-fsimx8mm-V<v>.tar.bz2

This is the main release itself containing all sources, the binary images, the documentation and the toolchain.

3.3 Release Content

These tar archives are compressed with bzip2. To see the files, you first have to unpack the archives

```
tar -xvf freertos-sdk-25.03-<arch>-<v>.tar.bz2
```

This will create a directory `<arch>-<v>` that contains all the files of the release. They often use a common naming scheme:

`<package>-<platform>-<v>.<extension>`

With the following meaning:

`<package>` The name of the package (e.g. `freertos-sdk`). If it is a source package, we also add the version number of the

	original package that our release is based on, for example `freertos-sdk-2.9.1`
`<platform>`	The name of a board, if the package is only valid on one board (e.g. `PicoCoreMX8MM`); or the name of an architecture, if the package is valid on different boards of the same architecture (e.g. `fsimx8mm`), or the string `f+s` or `fus` if the package is architecture independent.
`<v>`	Release version, consisting of a letter `V` for version and the year and month of the release (e.g. `V2021.07`).
`<extension>`	The extension of the package (e.g. `.bin`, `.tar.bz2`, etc.).

The following table lists the files that you get after unpacking the release archive. To avoid having a too excessive list, we use the wildcard `*` in some entries to refer to a whole group of similar file names that only differ in the name of the board or module.

Directory/File	Description
/	**Top directory**
Readme-freertos-f+s.txt	Release information (FreeRTOS)
setup-freertos	Script to unpack FreeRTOS source packages to a build directory
binaries/	**Images to be used with the board directly**
`*.bin, *.elf`	Precompiled examples for supported boards.
sources/	**Source packages**
`freertos-sdk-25.03-fsimx8mm-V2026.01.tar.bz2`	FreeRTOS source
toolchain/	**Cross-compilation toolchain**

Directory/File	Description
`gcc-arm-none-eabi-10.3-2021.10-x86_64-linux`	ARM toolchain to use with `<arch>`
`doc/`	**Documentation**
`FreeRTOS_on_FSiMX8MM_Boards_eng.pdf`	Manual on how to use/configuring the software
`patches/`	**Patches for Linux**
`0001-Improve-support-for-FreeRTOS-on-fsimx8mm-boards.patch`	Patch for Linux

Table 1: Content of the created release directory

3.4 Unpacking the Source Code

The source code packages are located in the `sources` subdirectory of the release archive. We will assume that you want to create a separate build directory where you extract the source code and build all the software.

We have prepared a shell script called `setup-freertos` that does this installation automatically. Just call it when you are in the top directory of the release and give the name of the build directory as argument.

```
cd <release-dir>
./setup-freertos <build-dir>
```

Add option `--dry-run` if you want to check first what this command will do. Then only a list of actions will be output but no actual changes will take place. For further information simply call

```
./setup-freertos --help
```

If you prefer to do the installation by hand, well, the script more or less executes the following commands, just with some more checks and directory switching.

```
mkdir <build-dir>
tar -xf freertos-sdk-25.03.00-fsimx8mm-<v>.tar.bz2
```

4 Getting started

4.1 Configure your host computer

In order to configure your computer properly (in order to use F&S software) please refer to the “AdvicesForLinuxOnPC” guide provided by F&S Elektronik Systeme. After you have done this, continue with this guide.

4.2 Get the tools and packages

Get the F&S FreeRTOS_BSP-package from the F&S website under

```
my F&S /picocoremx8mm/Linux/ FreeRTOS/freertos-sdk-25.03.00-
fsimx8mm-V<YEAR>.<MONTH>.tar.bz2
```

4.3 Install Content

We have prepared a shell script called `setup-freertos.sh` that does this installation automatically. Just call it when you are in the top directory of the release and give the name of the build directory as argument.

```
cd <release-dir>
./setup-freertos
```

Add option `--dry-run` if you want to check first what this command will do. Then only a list of actions will be output but no actual changes will take place. For further information simply call

```
./setup-freertos --help
```

If you prefer to do the installation by hand, well, the script more or less executes the following commands, just with some more checks and directory switching.

```
mkdir <build-dir>
tar -xf freertos-sdk-25.03.00-fsimx8mm-V<year>.<month>.tar.bz2
```

4.4 Installation of the GCC embedded toolchain

The examples can be built with the GCC embedded toolchain (gcc-arm-none-eabi-10.3-2021-10), which can be found under developer.arm.com or the toolchain directory of the release archive.

Extract the content to your filesystem:

```
tar -xvjf gcc-arm-none-eabi-${version}.tar.bz2
```

where ${version} will be replaced by the corresponding version you've downloaded.

It is necessary to export the ARMGCC_DIR environment variable:

```
export ARMGCC_DIR=/usr/local/arm/gcc-arm-none-eabi-${version}
```

For a more convenient way you can add this command to the rc file of your favorite shell (e.g. zshrc, bashrc, etc.)

4.5 Patches

This release includes two necessary patches for the Linux kernel:

- Use 0001-Add-support_m4-for-picocoremx8mm.patch for fsimx8mm-2024.10
- Use 0002-Add-support_m4-for-osm8mm.patch for osm8mm-2024.10.1

For newer fsimx8mm releases, this patch is no longer required.
Please ensure you apply the correct patch according to the version you are using.

4.6 Description of the FreeRTOS directory structure

The following table describes the directory structure of the FreeRTOS BSP

/	Top Directory
bin	After you have run the build script the output binaries can be found here in their specific $boardname-directory.
CMSIS	Contains the Cortex Microcontroller Software Interface Standard (CMSIS) library.
components	Contains software components provided by NXP, such as peripheral drivers, utilities, and board-specific support code. These are modular building blocks that can be reused across different projects.
devices	Contains socket specific files and drivers.
docs	Contains the original documentation by NXP.
middleware	Contains middleware libraries and frameworks from NXP.
rtos	Contains the operating system freertos.
tools	Contains different tools needed for the building process.

not_ported_examples	Contains examples that have not been ported yet. This can have different reasons like missing sensors or hardware on the EVK. Some of them will be ported in the future. If you are interested in porting one of these examples please contact F&S Electronic Systeme.
boards/	Contains the SoC and board specific Cortex-M4 examples. The first level distinguishes between the different SoC-architectures. At the second level you will find the SoC specific examples. For the MX8MM-examples the board specific examples are located directly in the directory of each example. The examples are structured as follows:
cmsis_driver_examples	Contains examples that shows the usage of the Cortex Micro-controller Software Interface Standard (CMSIS)
demo_apps	Here you can find the applications which highlight certain key features of the ARM Cortex-M4 Core combined with FreeRTOS and bare metal.
driver_examples	You can find simple applications here which are intended to show peripheral drivers working in the bare metal environment. Because some of the examples use special onboard sensors, they did not get ported.
multicore_examples	Here you can find examples, which demonstrate the multicore communication via RPMsg.
rtos_examples	These examples show the usage of different FreeRTOS-specific functions.

Table 2: Description of the directory structure

5 Building the examples

To simplify the process of building, configuring the examples and cleaning up we provide you with a set of bash scripts located in the root directory of the FreeRTOS BSP:

5.1 build.sh

This script will build all examples for you. You need to select the board, the build type, and the build mode.

```
./build.sh
```

and follow the instructions:

```
Select your target board:
1) PICOCOREMX8MM
2) PICOCOREMX8MX
3) FS_8MM_OSM_SF
#? 1
Board selected: PICOCOREMX8MM (folder: fsimx8mm)
Select build type:
1) release
2) debug
#? 1
Build type selected: release
Build mode:
1) Single example
2) All examples
#? 1
Select an example:
1) /boards/fsimx8mm/cmsis_driver_examples/ecspi/int_loop-
back_transfer
2) /boards/fsimx8mm/cmsis_driver_examples/ecspi/sdma_loop-
back_transfer
...
```

5.2 clean_all.sh

This script can also clean all examples at once.

```
./clean_all.sh
```

6 Adding custom boards

If you are using a custom board, you must add it to the build.sh script to enable the build process and generate the required configuration files (or simply copy them from an existing board).

First, create a directory for your board's examples inside the boards/ directory.

```
mkdir boards/<dir_name>
```

Next, modify the following lines in build.sh:

```
declare -A BOARD_MAP=(
    ["PICOCOREMX8MM"]="fsimx8mm"
    ["PICOCOREMX8MX"]="fsimx8mm"
    ["FS_8MM_OSM_SF"]="fsimx8mm"
    ["boardname"]="dir_name"
)
```

boardname → the name of your board

dir_name → must exactly match the name of the directory inside boards/ where your examples are stored.

To build the examples, the armgcc directory must be included in your board's directory. You can copy an existing armgcc directory from another example and modify its CMakeLists.txt file to include your own example.

7 Execute the examples

The examples can be executed either from U-Boot using a .bin file or from Linux using an .elf (Executable and Linkable Format) file via the `remoteproc` and `rpmsg` frameworks.

7.1 Execute in U-Boot

F&S provides you with a modified U-Boot which can make use of the Cortex-M4 via the *bootaux* command. Since our U-Boot is heavily modified compared to the official release from NXP, it's not advisable to use any other than the one provided by F&S.

F&S added some environment variables to simplify the auxiliary core handling:

Set the name of the binary to be loaded:

```
setenv m4_file <example_name.bin>
```

<example_name.bin> is the name of the example to be loaded. For example: hello_world.bin

Define the `m4` command to load and start the firmware:

```
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
```

This command fetches the binary via TFTP, copies it to memory at address `0x007E0000`, and boots it using `bootaux`. Alternatively, the binaries can be loaded by mounting a storage medium, e.g. a USB stick as follows:

```
setenv m4 "usb start; fatload usb 0:1 $loadaddr ${m4_file}; cp.b
$loadaddr 0x007E0000 $filesize; bootaux 0x007E0000"
```

Start the M4 core by running:

```
run m4
```

This will initiate the TFTP transfer, copy the firmware, and start the M4 core. Once started, the firmware will begin executing. You should see output via the M4 UART.

7.2 Execute in Linux

To execute FreeRTOS examples from Linux, follow these steps:

Set U-Boot Environment Variable:

Before booting Linux, ensure the following environment variable is set in U-Boot:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

It ensures that the clock needed for the M4 core to operate is active when RemoteProc is used to start the M4 core from Linux.

Load the example over tftp server:

```
tftp -g -r <example_name.elf> -l /lib/firmware/<example_name.elf>
<tftp_server_ip>
```

<example_name.elf> is the name of the ELF example to be loaded.
For example: hello_world_cm4.elf
Alternatively, can be load by mounting a storage medium as follows.

```
mount /dev/<storage_dev> /mnt/
cp /mnt/<example_name.elf> /lib/firmware/
```

Set the firmware for the remote processor:

```
echo <example_name.elf> > /sys/class/remoteproc/re-
moteproc0/firmware
```

<example_name.elf> is the name of the ELF example to be loaded.
For example: hello_world_cm4.elf
This tells the remoteproc framework which firmware should be loaded the next time the remote processor is started.

Start the Cortex-M4 Core:

```
echo start > /sys/class/remoteproc/remoteproc0/state
```

If the firmware is successfully started, you should see corresponding UART output from the M4 side.

Stop the Cortex-M4 Core:

```
echo stop > /sys/class/remoteproc/remoteproc0/state
```

To load another example, first stop the remoteproc framework.

7.3 Execute using a SEGGER J-Link Debugger

It is possible to use a J-Link debugger to run and debug M4 applications. This requires Visual Studio Code with some configuration. Currently, this feature is not supported on our development Machine, since the necessary Visual Studio Code Extension is not available on Fedora.

Configuration

At first you need to install the MCUXpresso Extension in Visual Studio Code.

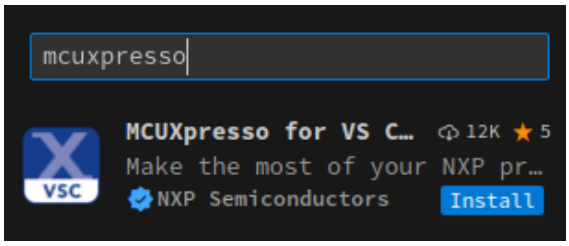


Figure 3: MCUXpresso Extension

When the extension is installed, you need some more Software. You can install this through the extension. Open the Extension through the sidebar of VS Code and select the option “Open MCUXpresso installer”. This can be found in the quickstart panel.

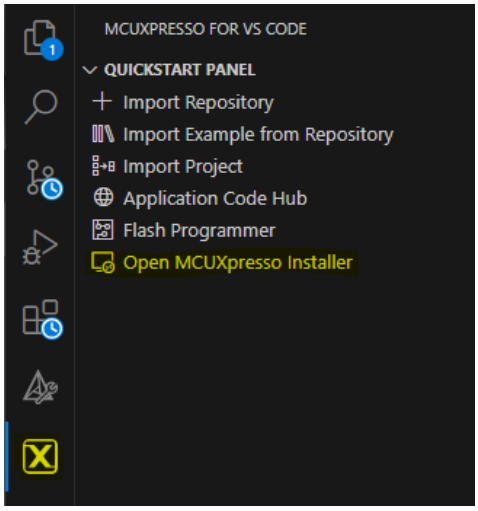


Figure 4: Quickstart Panel

Execute the examples

It will open another Window. There, make sure to select:

- MCUXpresso SDK Developer
- SEGGER J-Link
- Arm GNU Toolchain

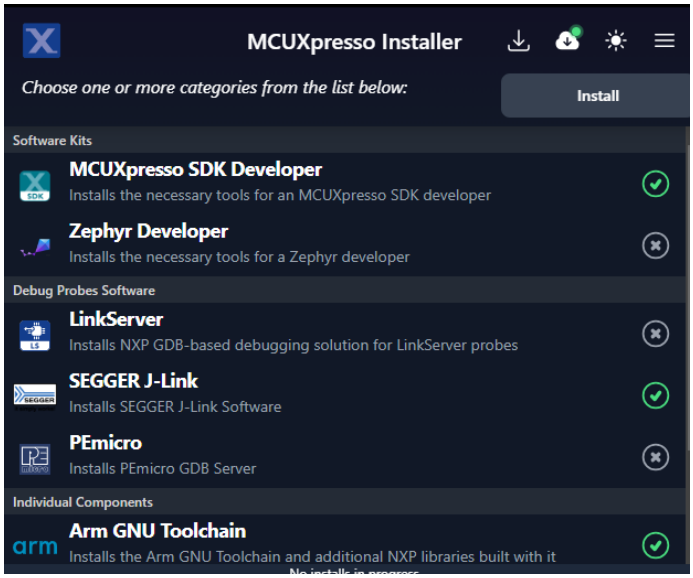


Figure 5: Device tree entry

This will install the the NXP tools, the ARM toolchain, and the debugger software to your windows host.

The debugger Software needs Information about the processor that is not provided by the debugger Software itself. NXP released a Patch to add the Information. Please Download it here.

The Archive includes an directory called JLink. It has the same structure as the JLink Software. Add the files of the Archive to your installation, according to the directory structure. Your installation is usually saved at C:\Programs\SEGGER.

Import the repository

In VSCode Click on the MCUXpresso Symbol in your extension bar. There, in the Quickstart Panel Click: Import Repository. A Context Window will appear.

Select local and set the location to the sdk and click import. Now the sdk should be shown under projects.

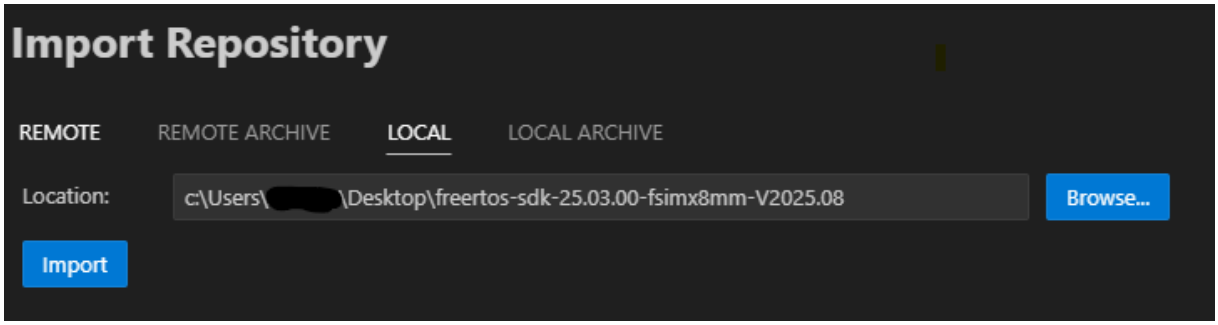


Figure 6: Importing a repository

Import the example

Now you can Import your examples. For this, select Import example from repository in the quickstart panel. In the Context Window fill the values out. Most are the only possible selection. Choose the example and click create.

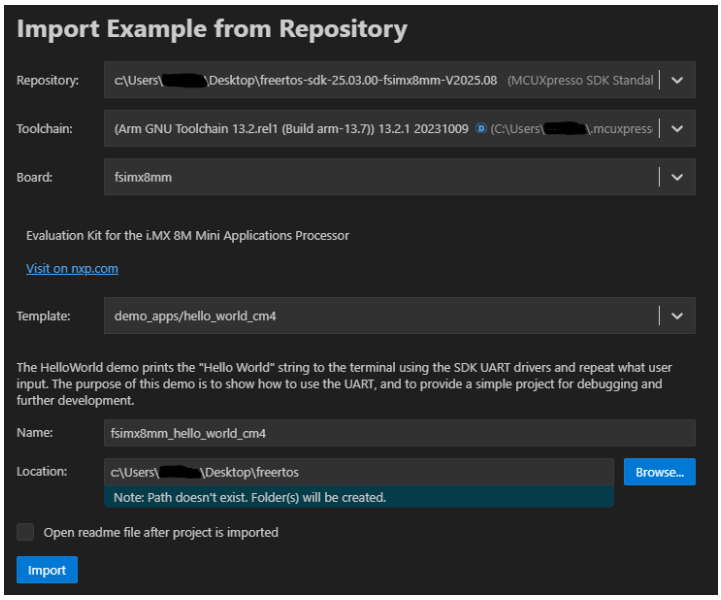


Figure 7: Importing an example

Since the F&S Release supports multiple boards, you need to specify which board you want to build for. By default, the PICOCOREMX8MM is selected. To use a different board, open the CMakeLists.txt file in your project directory (*/armgcc/CMakeLists.txt) and change the -DPICOCORE8MM definition to either -DPICOCOREMX8MX or -DFS_8MM_OSM_SF.

```
# add Board definitions (PICOCOREMX8MM - PICOCOREMX8MX - FS_8MM_OSM_SF)
add_definitions(-DPICOCOREMX8MM)
```

Figure 8: Target Board selection

After saving the changes, CMake will automatically apply them. To update all examples at once, you can use the Replace tool in Visual Studio Code.

Build and run

To run the example on your board connect the debugger with your Board and PC and Power the Board on. Go back to the MCUXpresso extension. In the projects tab you will see your example listed.

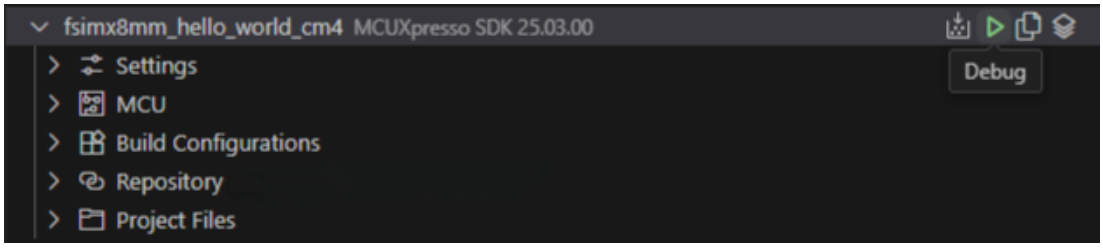


Figure 9: Running the example

By clicking the play button on the right you can build and run your application. The button left of it allows to only build your example. After starting your application you can make use of debugging features like breakpoints or variable analysis.

8 FreeRTOS examples

In this chapter we will provide you with necessary information on the demo and driver applications.

The “**Description**” will inform you about the demo's purpose.

In the “**Modifications made**” section you will find useful information if changes were made to certain files by F&S and the reason behind these changes.

“**Changes needed**” is the most important section. You will find the information necessary to successfully build and execute the examples here.

The last section, “**Execute binary**” will tell you the required steps to execute the image built.

8.1 General build and run information

For PicoCoreMX8MM/MX connect UART4 (Cortex M4) and UART1 (Cortex A53) with two serial cables (serial-to-modem) to your PC.

For FS 8MM OSM-SF connect UART3 (Cortex M4) and UART1 (Cortex A53) with two serial cables (serial-to-modem) to your PC.

Open up two Terminals and connect the UARTs via the COM interface and the following settings:

```
Baud rate: 115200
Data: 8 bit
Parity: none
Stop: 1 bit
Flow control: none
Transmit delay: 0 msec/char 0 msec/line
```

Build the examples like described in **Building the examples** and copy them to you tftp-directory.

To test the multicores examples or to start the examples from Linux, the device tree must be modified. This ensures that the Cortex-M4 core has access to its required peripherals preventing resource conflicts with the Linux kernel.

To enable the corresponding firmware support, uncomment the #define SUPPORT_M4:

```
16   /* Activate this if you want to use the Cortex-M4 core */
17   //#define SUPPORT_M4
```

Figure 10: Device tree SUPPORT_M4

8.2 Cmsis_driver_examples

8.2.1 ecspi

Int_loopback_transfer

Description

CMSIS-Driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to http://www.keil.com/pack/doc/cmsis/Driver/html/index.html.

The cmsis_ecspi_int_loopback_transfer example shows how to use CMSIS ECSPI driver in interrupt way:

In this example , ECSPI will do a loopback transfer in interrupt way, so, there is no need to set up any pins. And we should set the ECSPIx->TESTREG[LBC] bit, this bit is used in Master mode only. When this bit is set, the ECSPI connects the transmitter and receiver sections internally, and the data shifted out from the most-significant bit of the shift register is looped back into the least-significant bit of the Shift register. In this way, a self-test of the complete transmit/receive path can be made. The output pins are not affected, and the input pins are ignored.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None.

Example Execution

This example can be executed from U-Boot or Linux.

1. Execute from U-Boot

```
setenv m4_file "cmsis_ecspi_int_loopback_transfer.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r cmsis_ecspi_int_loopback_transfer_cm4.elf -l
/lib/firmware/cmsis_ecspi_int_loopback_transfer_cm4.elf
<tftp_server_ip>

echo cmsis_ecspi_int_loopback_transfer_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
This is ECSPI CMSIS interrupt loopback transfer example.

The ECSPI will connect the transmitter and receiver sections in-
ternally.

Start transfer...


  This is ECSPI_MasterSignalEvent_t.


Transfer completed!

ECSPI transfer all data matched!
```

sdma_loopback_transfer

Description

CMSIS-Driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to http://www.keil.com/pack/doc/cmsis/Driver/html/index.html.

The cmsis_ecspi_sdma_loopback_transfer example shows how to use CMSIS ECSPI driver in SDMA way:

In this example, ECSPI will do a loopback transfer in SDMA way, so, there is no need to set up any pins. And we should set the ECSPIx->TESTREG[LBC] bit, this bit is used in Master mode only. When this bit is set, the ECSPI connects the transmitter and receiver sections internally, and the data shifted out from the most-significant bit of the shift register is looped back into the least-significant bit of the Shift register.

In this way, a self-test of the complete transmit/receive path can be made. The output pins are not affected,

and the input pins are ignored.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

To use the example, please mind the necessary changes.

Example Execution

This example can be executed from U-Boot or Linux.

1. Execute from U-Boot

```
setenv m4_file "cmsis_ecspi_sdma_loopback_transfer.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r cmsis_ecspi_sdma_loopback_transfer_cm4.elf -l
/lib/firmware/cmsis_ecspi_sdma_loopback_transfer_cm4.elf
<tftp_server_ip>

echo cmsis_ecspi_sdma_loopback_transfer_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

The log below shows the output of the hello world demo in the terminal window:

```
This is ECSPI CMSIS SDMA loopback transfer example.

The ECSPI will connect the transmitter and receiver sections in-
ternally.

Start transfer...


  This is ECSPI_MasterSignalEvent_t


Transfer completed!

ECSPI transfer all data matched!
```

8.2.2 i2c

int_b2b_transfer / Master

Description

CMSIS-Driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to http://www.keil.com/pack/doc/cmsis/Driver/html/index.html.

The i2c_interrupt_b2b_transfer_master example shows how to use CMSIS i2c driver as master to do board to board transfer with interrupt:

In this example, one i2c instance as master and another i2c instance on the other board as slave. Master sends a piece of data to slave, and receive a piece of data from slave. This example checks if the data received from slave is correct.

Modifications made

PCOREMX8MM: Changed I2C port to I2C1

PCOREMX8MX: Changed I2C port to I2C4

OSM8MM: Changed I2C port to I2C2

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
I2C_A_SCL	J1_4		J11_16
I2C_A_SDA	J1_6		J11_17
GND	---		J11_11

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.3
I2C_A_SCL	J1I_AA15	J1_4	J11_16
I2C_A_SDA	J1I_AA15	J1_6	J11_17
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux. To successfully run this example, first start the int_b2b_transfer_slave example on the other board

1. Execute from U-Boot

```
setenv m4_file "cmsis_ii2c_interrupt_b2b_transfer_master.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r cmsis_ii2c_interrupt_b2b_transfer_master_cm4.elf -l
/lib/firmware/cmsis_ii2c_interrupt_b2b_transfer_master_cm4.elf
<tftp_server_ip>
echo cmsis_ii2c_interrupt_b2b_transfer_master_cm4.elf >
/sys/class/remoteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the following message is displayed in the terminal:

```
CMSIS I2C board2board interrupt example -- Master transfer.
Master will send data :
0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7
0x 8  0x 9  0x a  0x b  0x c  0x d  0x e  0x f
0x10  0x11  0x12  0x13  0x14  0x15  0x16  0x17
0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e  0x1f

Receive sent data from slave :
0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7
0x 8  0x 9  0x a  0x b  0x c  0x d  0x e  0x f
0x10  0x11  0x12  0x13  0x14  0x15  0x16  0x17
0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e  0x1f


End of I2C example .
```

int_b2b_transfer / Slave

Description

CMSIS-Driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to http://www.keil.com/pack/doc/cmsis/Driver/html/index.html.

The i2c_interrupt_b2b_transfer_master example shows how to use CMSIS i2c driver as master to do board to board transfer with interrupt:

In this example, one i2c instance as master and another i2c instance on the other board as slave. Master sends a piece of data to slave, and receive a piece of data from slave. This example checks if the data received from slave is correct.

Modifications made

PCOREMX8MM: Changed I2C port to I2C1

PCOREMX8MX: Changed I2C port to I2C4

OSM8MM: Changed I2C port to I2C2

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
I2C_A_SCL	J1_4		J11_16
I2C_A_SDA	J1_6		J11_17
GND	---		J11_11

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.3
I2C_A_SCL	J1I_AA15	J1_4	J11_16
I2C_A_SDA	J1I_AA15	J1_6	J11_17
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux. To successfully run this example, start the int_b2b_transfer_master example on the other board afterwards.

1. Execute from U-Boot

```
setenv m4_file "cmsis_ii2c_interrupt_b2b_transfer_slave.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r cmsis_ii2c_interrupt_b2b_transfer_slave_cm4.elf -l
/lib/firmware/cmsis_ii2c_interrupt_b2b_transfer_slave_cm4.elf
<tftp_server_ip>

echo cmsis_ii2c_interrupt_b2b_transfer_slave_cm4.elf >
/sys/class/remoteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the following message is displayed in the terminal:

```
CMSIS I2C board2board interrupt example -- Slave transfer.



Slave received data :
0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7
0x 8  0x 9  0x a  0x b  0x c  0x d  0x e  0x f
0x10  0x11  0x12  0x13  0x14  0x15  0x16  0x17
0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e  0x1f



End of I2C example .
```

8.2.3 uart

cmsis_iuart_interrupt_transfer

Description

CMSIS-Driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to http://www.keil.com/pack/doc/cmsis/Driver/html/index.html.

The cmsis_uart_interrupt_transfer example shows how to use uart cmsis driver in interrupt way:

In this example, one uart instance connect to PC through uart, the board will send back all characters that PC send to the board.

The example echo every 8 characters, so input 8 characters every time.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv m4_file "cmsis_iuart_interrupt_transfer.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r cmsis_iuart_interrupt_transfer_cm4.elf -l
/lib/firmware/cmsis_iuart_interrupt_transfer_cm4.elf
<tftp_server_ip>

echo cmsis_iuart_interrupt_transfer_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
USART CMSIS interrupt example

Board receives 8 characters then sends them out

Now please input:
```

cmsis_iuart_sdma_transfer

Description

CMSIS-Driver defines generic peripheral driver interfaces for middleware making it reusable across a wide range of supported microcontroller devices. The API connects microcontroller peripherals with middleware that implements for example communication stacks, file systems, or graphic user interfaces. More information and usage method please refer to http://www.keil.com/pack/doc/cmsis/Driver/html/index.html.

The cmsis_uart_sdma_transfer example shows how to use uart cmsis driver with SDMA:

In this example, one uart instance connect to PC through uart, the board will send back all characters that PC send to the board.

The example echo every 8 characters, so input 8 characters every time.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

To use the example, please mind the necessary changes.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv m4_file "cmsis_iuart_sdma_transfer.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r cmsis_iuart_sdma_transfer_cm4.elf -l /lib/firmware/cm-
sis_iuart_sdma_transfer_cm4.elf <tftp_server_ip>
echo cmsis_iuart_sdma_transfer_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
USART CMSIS SDMA example
Board receives 8 characters then sends them out
Now please input:
```

8.3 demo_apps

Remark

The documentation is based on the FreeRTOS BSP 2.9.1 package from NXP.

Some of the software examples provided by NXP expect a certain module or sensor to be available on the board. Since F&S boards do NOT provide these, the associated examples weren't ported at all.

8.3.1 hello_world

Description

The Hello World demo application provides a sanity check for the new SDK build environments and board bring up. The Hello World demo prints the "Hello World" string to the terminal using the SDK UART drivers. The purpose of this demo is to show how to use the UART, and to provide a simple project for debugging and further development.

Please input one character at a time. If you input too many characters each time, the receiver may overflow because the low level UART uses simple polling way for receiving. If you want to try inputting many characters each time, just define DEBUG_CONSOLE_TRANSFER_NON_BLOCKING in your project to use the advanced debug console utility.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv m4_file "hello_world.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r hello_world_cm4.elf -l
/lib/firmware/hello_world_cm4.elf <tftp_server_ip>

echo hello_world_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

The log below shows the output of the hello world demo in the terminal window:

```
hello world.
```

8.3.2 sai_low_power_audio

This example has not been ported yet.

8.4 driver_examples

8.4.1 ecspi

ecspi_loopback

Description

The ecspi_loopback demo shows how the ecspi do a loopback transfer internally. The ECSPI connects the transmitter and receiver sections internally, and the data shifted out from the most-significant bit of the shift register is looped back into the least-significant bit of the Shift register. In this way, a self-test of the complete transmit/receive path can be made. The ouput pins are not affected, and the input pins are ignored.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv m4_file "ecspi_loopback.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r ecspi_loopback_cm4.elf -l /lib/firmware/ecspi_loop-
back_cm4.elf <tftp_server_ip>

echo ecspi_loopback_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

If the demo run successfully, the below log will be print in the terminal window:

```
***ECSPI Loopback Demo***


This demo is a loopback transfer test for ECSPI.

The ECSPI will connect the transmitter and receiver sections in-
ternally.

So, there is no need to connect the MOSI and MISO pins.


ECSPI loopback test pass!
```

interrupt_b2b_transfer / Master

Description

The ecspi_interrupt_b2b_transfer example shows how to use ECSPI driver in interrupt way:

In this example , we need two boards, one board used as ECSPI master and another board used as ECSPI slave. The file 'ecspi_interrupt_b2b_transfer_master.c' includes the ECSPI master code. This example uses the transactional API in ECSPI driver.

1. ECSPI master send/received data to/from ECSPI slave in interrupt. (ECSPI Slave using interrupt to receive/send the data)

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1_58		J11_5
SPI_B_MOSI	J1_60		J11_6
SPI_B_SCLK	J1_62		J11_3
SPI_B_SS0	J1_64		J11_4
GND	---		J11_11

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1I_Y22	J1_58	J11_5
SPI_B_MOSI	J1I_Y23	J1_60	J11_6
SPI_B_SCLK	J1I_Y21	J1_62	J11_3
SPI_B_SS0	J1I_AA23	J1_64	J11_4
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux. To successfully run this example, first start the interrupt_b2b_transfer_slave example on the other board.

1. Execute from U-Boot

```
setenv m4_file "ecspi_interrupt_b2b_transfer_master.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r ecspi_interrupt_b2b_transfer_master_cm4.elf -l
/lib/firmware/ecspi_interrupt_b2b_transfer_master_cm4.elf
<tftp_server_ip>
echo ecspi_interrupt_b2b_transfer_master_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
ECSPI board to board interrupt example.
This example use one board as master and another as slave.
Master and slave uses interrupt way. Slave should start first.
Please make sure you make the correct line connection. Basically,
the connection is:
ECSPI_master -- ECSPI_slave
   CLK      --    CLK
   PCS      --    PCS
   MOSI     --    MOSI
   MISO     --    MISO
   GND      --    GND

 Master transmit:

  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F 10
 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30
 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40

ECSPI transfer all data matched!

 Master received:

  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F 10
 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30
 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40

 Press any key to run again
```

interrupt_b2b_transfer / Slave

Description

The ecspi_interrupt_b2b_transfer example shows how to use ECSPI driver in interrupt way:

In this example, we need two boards, one board used as ECSPI master and another board used as ECSPI slave. The file 'ecspi_interrupt_b2b_transfer_slave.c' includes the ECSPI slave code. This example uses the transactional API in ECSPI driver.

1. ECSPI master send/received data to/from ECSPI slave in interrupt. (ECSPI Slave using interrupt to receive/send the data)

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1_58		J11_5
SPI_B_MOSI	J1_60		J11_6
SPI_B_SCLK	J1_62		J11_3
SPI_B_SS0	J1_64		J11_4
GND	---		J11_11

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1I_Y22	J1_58	J11_5
SPI_B_MOSI	J1I_Y23	J1_60	J11_6
SPI_B_SCLK	J1I_Y21	J1_62	J11_3
SPI_B_SS0	J1I_AA23	J1_64	J11_4
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux. To successfully run this example, start the interrupt_b2b_transfer_master example on the other board afterwards.

1. Execute from U-Boot

```
setenv "ecspi_interrupt_b2b_transfer_slave.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r ecspi_interrupt_b2b_transfer_slave_cm4.elf -l
/lib/firmware/ecspi_interrupt_b2b_transfer_slave_cm4.elf
<tftp_server_ip>
echo ecspi_interrupt_b2b_transfer_slave_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
ECSPI board to board interrupt example.

 Slave example is running...

 Slave starts to receive data!
 This is ECSPI slave transfer completed callback.
 It's a successful transfer.

 Slave starts to transmit data!
 This is ECSPI slave transfer completed callback.
 It's a successful transfer.

 Slave receive:
      1  2  3  4  5  6  7  8  9  A  B  C  D  E  F 10
     11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
     21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30
     31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40

 Slave example is running...
```

polling_b2b_transfer / Master

Description

The ecspi_polling_b2b_transfer example shows how to use ECSPI driver in polling way:

In this example, we need two boards, one board used as ECSPI master and another board used as ECSPI slave. The file 'ecspi_polling_b2b_transfer_master.c' includes the ECSPI master code.

1. ECSPI master send/receive data to/from ECSPI slave in polling. (ECSPI Slave using interrupt to receive/send the data)

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1_58		J11_5
SPI_B_MOSI	J1_60		J11_6
SPI_B_SCLK	J1_62		J11_3
SPI_B_SS0	J1_64		J11_4
GND	---		J11_11

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1I_Y22	J1_58	J11_5
SPI_B_MOSI	J1I_Y23	J1_60	J11_6
SPI_B_SCLK	J1I_Y21	J1_62	J11_3
SPI_B_SS0	J1I_AA23	J1_64	J11_4
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux. To successfully run this example, first start the polling_b2b_transfer_slave example on the other board.

1. Execute from U-Boot

```
setenv "ecspi_polling_b2b_transfer_master.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r ecspi_polling_b2b_transfer_master_cm4.elf -l
/lib/firmware/ecspi_polling_b2b_transfer_master_cm4.elf
<tftp_server_ip>
echo ecspi_polling_b2b_transfer_master_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output
If the demo runs successfully, the log below will be print in the terminal window:

```
ECSPI board to board polling example.
This example use one board as master and another as slave.
Master uses polling way and slave uses interrupt way.
Please make sure you make the correct line connection. Basically,
the connection is:
ECSPI_master -- ECSPI_slave
   CLK      --    CLK
   PCS      --    PCS
   MOSI     --    MOSI
   MISO     --    MISO
   GND      --    GND

 Master transmit:

  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F 10
 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30
 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40

ECSPI transfer all data matched!

 Master received:

  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F 10
 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30
 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40

 Press any key to run again
```

polling_b2b_transfer / Slave

Description

The ecspi_polling_b2b_transfer example shows how to use ECSPI driver in polling way:

In this example, we need two boards, one board used as ECSPI master and another board used as ECSPI slave. The file 'ecspi_polling_b2b_transfer_slave.c' includes the ECSPI slave code.

1. ECSPI master send/received data to/from ECSPI slave in polling . (ECSPI Slave using interrupt to receive/send the data)

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1_58		J11_5
SPI_B_MOSI	J1_60		J11_6
SPI_B_SCLK	J1_62		J11_3
SPI_B_SS0	J1_64		J11_4
GND	---		J11_11

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.4
SPI_B_MISO	J1I_Y22	J1_58	J11_5
SPI_B_MOSI	J1I_Y23	J1_60	J11_6
SPI_B_SCLK	J1I_Y21	J1_62	J11_3
SPI_B_SS0	J1I_AA23	J1_64	J11_4
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux. To successfully run this example, start the polling_b2b_transfer_master example on the other board afterwards.

1. Execute from U-Boot

```
setenv "ecspi_polling_b2b_transfer_slave.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r ecspi_polling_b2b_transfer_slave_cm4.elf -l
/lib/firmware/ecspi_polling_b2b_transfer_slave_cm4.elf
<tftp_server_ip>
echo ecspi_polling_b2b_transfer_slave_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output
When the demo runs successfully, the log would be seen on the debug terminal like:

```
ECSPI board to board polling example.

 Slave example is running...

 Slave starts to receive data!
 This is ECSPI slave transfer completed callback.
 It's a successful transfer.

 Slave starts to transmit data!
 This is ECSPI slave transfer completed callback.
 It's a successful transfer.

 Slave receive:
      1  2  3  4  5  6  7  8  9  A  B  C  D  E  F 10
     11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20
     21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F 30
     31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40

 Slave example is running...
```

8.4.2 gpio

led_output

Description

The GPIO Example project is a demonstration program that uses the KSDK software to manipulate the general-purpose outputs. The example is supported by the set, clear, and toggle write-only registers for each port output data register. The example take turns to shine the LED.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
SPI_B_MOSI / LED	J1_60		J11_6

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.4
SPI_B_MOSI / LED	J1I_Y23	J1_60	J11_6

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "igpio_led_output.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r igpio_led_output_cm4.elf -l /lib/firmware/ig-
pio_led_output_cm4.elf <tftp_server_ip>

echo igpio_led_output_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the example runs successfully, the following message is displayed in the terminal:

```
GPIO Driver example

 The LED is blinking.
```

8.4.3 gpt

gpt capture

The example can't be ported at the moment because no reasonable pin is available.

gpt_timer

Description

The gpt_timer project is a simple demonstration program of the SDK GPT driver. It sets up the GPT hardware block to trigger a periodic interrupt after every 1 second. When the GPT interrupt is triggered a message a printed on the UART terminal.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "gpt_timer.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r gpt_timer_cm4.elf -l /lib/firmware/gpt_timer_cm4.elf
<tftp_server_ip>
echo gpt_timer_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the example runs successfully, the following message is displayed in the terminal:

```
Press any key to start the example
s
Starting GPT timer ...
 GPT interrupt is occurred !
 GPT interrupt is occurred !
 GPT interrupt is occurred !
 GPT interrupt is occurred !
 .
 .
 .
 GPT interrupt is occurred !
```

8.4.4 i2c

interrupt_b2b_transfer / Master

Description

The i2c_interrupt_b2b_transfer_master example shows how to use i2c driver as master to do board to board transfer with interrupt:

In this example, one i2c instance acts as the master and another i2c instance on the other board as slave. The master sends a piece of data to the slave, and receives a piece of data from the slave. This example checks if the data received from the slave is correct.

Modifications made

PCOREMX8MM: Changed I2C port to I2C1

PCOREMX8MX: Changed I2C port to I2C4

OSM8MM: Changed I2C port to I2C2

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed:

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
I2C_A_SCL	J1_4		J11_16
I2C_A_SDA	J1_6		J11_17
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux. To successfully run this example, first start the i2c_interrupt_b2b_transfer_slave example on the other board.

1. Execute from U-Boot

```
setenv "ii2c_interrupt_b2b_transfer_master.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r ii2c_interrupt_b2b_transfer_master_cm4.elf -l
/lib/firmware/ii2c_interrupt_b2b_transfer_master_cm4.elf
<tftp_server_ip>

echo ii2c_interrupt_b2b_transfer_master_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the example runs successfully, following information can be seen on the terminal:

```
When the demo runs successfully, the following message is dis-
played in the terminal:


I2C board2board interrupt example -- Master transfer.

Master will send data :

0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7

0x 8  0x 9  0x a  0x b  0x c  0x d  0x e  0x f

0x10  0x11  0x12  0x13  0x14  0x15  0x16  0x17

0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e  0x1f


Receive sent data from slave :

0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7

0x 8  0x 9  0x a  0x b  0x c  0x d  0x e  0x f

0x10  0x11  0x12  0x13  0x14  0x15  0x16  0x17

0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e  0x1f



End of I2C example .
```

interrupt_b2b_transfer / Slave

Description

The ii2c_interrupt_b2b_transfer_slave example shows how to use i2c driver as slave to do board to board transfer with interrupt:

In this example, one i2c instance as slave and another i2c instance on the other board as master. Master sends a piece of data to slave, and receive a piece of data from slave. This example checks if the data received from slave is correct.

Modifications made

PCOREMX8MM: Changed I2C port to I2C1

PCOREMX8MX: Changed I2C port to I2C4

OSM8MM: Changed I2C port to I2C2

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
I2C_A_SCL	J1_4		J11_16
I2C_A_SDA	J1_6		J11_17
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux. To successfully run this example, start the i2c_interrupt_b2b_transfer_master example on the other board afterwards.

1. Execute from U-Boot

```
setenv "ii2c_interrupt_b2b_transfer_slave.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r ii2c_interrupt_b2b_transfer_slave_cm4.elf -l
/lib/firmware/ii2c_interrupt_b2b_transfer_slave_cm4.elf
<tftp_server_ip>

echo ii2c_interrupt_b2b_transfer_slave_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the following message is displayed in the terminal:

```
I2C board2board interrupt example -- Slave transfer.



Slave received data :
0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7
0x 8  0x 9  0x a  0x b  0x c  0x d  0x e  0x f
0x10  0x11  0x12  0x13  0x14  0x15  0x16  0x17
0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e  0x1f



End of I2C example .
```

polling_b2b_transfer / Master

Description

The ii2c_polling_b2b_transfer_master example shows how to use i2c driver as master to do board to board transfer using polling method:

In this example, one i2c instance as master and another i2c instance on the other board as slave. Master sends a piece of data to slave, and receive a piece of data from slave. This example checks if the data received from slave is correct.

Modifications made

PCOREMX8MM: Changed I2C port to I2C1

PCOREMX8MX: Changed I2C port to I2C4

OSM8MM: Changed I2C port to I2C2

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
I2C_A_SCL	J1_4		J11_16
I2C_A_SDA	J1_6		J11_17
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux To successfully run this example, first start the i2c_polling_b2b_transfer_slave example on the other board.

1. Execute from U-Boot

```
setenv "ii2c_polling_b2b_transfer_master.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r ii2c_polling_b2b_transfer_master_cm4.elf -l
/lib/firmware/ii2c_polling_b2b_transfer_master_cm4.elf
<tftp_server_ip>

echo ii2c_polling_b2b_transfer_master_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the following message is displayed in the terminal:

```
I2C board2board polling example -- Master transfer.
Master will send data :
0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7
0x 8  0x 9  0x a  0x b  0x c  0x d  0x e  0x f
0x10  0x11  0x12  0x13  0x14  0x15  0x16  0x17
0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e  0x1f

Receive sent data from slave :
0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7
0x 8  0x 9  0x a  0x b  0x c  0x d  0x e  0x f
0x10  0x11  0x12  0x13  0x14  0x15  0x16  0x17
0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e  0x1f


End of I2C example .
```

polling_b2b_transfer / Slave

Description

The i2c_polling_b2b_transfer_slave example shows how to use i2c driver as slave to do board to board transfer with a polling master:

In this example, one i2c instance as slave and another i2c instance on the other board as master. Master sends a piece of data to slave, and receive a piece of data from slave. This example checks if the data received from slave is correct.

Modifications made

PCOREMX8MM: Changed I2C port to I2C1

PCOREMX8MX: Changed I2C port to I2C4

OSM8MM: Changed I2C port to I2C2

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
I2C_A_SCL	J1_4		J11_16
I2C_A_SDA	J1_6		J11_17
GND	---		J11_11

Example Execution

This example can be executed from U-Boot or in Linux. To successfully run this example, start the i2c_polling_b2b_transfer_master example on the other board afterwards.

1. Execute from U-Boot

```
setenv "ii2c_polling_b2b_transfer_slave.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r ii2c_polling_b2b_transfer_slave_cm4.elf -l
/lib/firmware/ii2c_polling_b2b_transfer_slave_cm4.elf
<tftp_server_ip>

echo ii2c_polling_b2b_transfer_slave_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the following message is displayed in the terminal:

```
I2C board2board polling example -- Slave transfer.



Slave received data :
0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7
0x 8  0x 9  0x a  0x b  0x c  0x d  0x e  0x f
0x10  0x11  0x12  0x13  0x14  0x15  0x16  0x17
0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e  0x1f



End of I2C example .
```

8.4.5 pdm

Hasn't been ported yet.

8.4.6 pwm

Description

The PWM example shows how to setup and generate a PWM signal. The frequency and duty cycle can be programmed. When booting the A53 and using the default configuration set by the function PWM_GetDefaultConfig(), the signal will be gated off by the Power State Coordination Interface (PSCI) to save energy. To counter this behavior some modifications have been made to keep the signal active.

Modifications made

pwm.c:

```
    pwmConfig.enableStopMode = true;
    pwmConfig.enableDozeMode = true;
    pwmConfig.enableWaitMode = true;
    pwmConfig.enableDebugMode = true;
```

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
PWM	J2_63		J11_34

Function	OSM Rev 1.2	ADP-OSM-BB Rev 1.2	BBDSI Rev 1.3
PWM	J3J_F18	J2_63	J11_34

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "ipwm.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r ipwm_cm4.elf -l /lib/firmware/ipwm_cm4.elf
<tftp_server_ip>
echo ipwm_cm4.elf > /sys/class/remoteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

The following message is displayed in the terminal window:

```
PWM driver example
```

On an oscilloscope you should see clear rectangular pulses. Alternatively, a LED can be used. It should visibly flash with a relatively high frequency.

8.4.7 rdc

Description

The RDC example shows how to control the peripheral and memory region access policy using RDC and RDC_SEMA42.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000.
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "rdc.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r rdc_cm4.elf -l /lib/firmware/rdc_cm4.elf
<tftp_server_ip>
echo rdc_cm4.elf > /sys/class/remoteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

The log below is shown in the terminal window:

```
RDC Example:
RDC Peripheral access control
RDC Peripheral access control with SEMA42
RDC memory region access control

RDC Example Succes
```

8.4.8 sai

interrupt_transfer

Hasn’t been ported yet.

sdma_transfer

Hasn’t been ported yet.

8.4.9 sdma

memory_to_memory

Description

The EDMA memory to memory example is a simple demonstration program that uses the SDK software. It executes one shot transfer from source buffer to destination buffer using the SDK EDMA drivers. The purpose of this example is to show how to use the EDMA and to provide a simple example for debugging and further development.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

To use the example, please mind the necessary changes.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "sdma_memory_to_memory.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r sdma_memory_to_memory_cm4.elf -l
/lib/firmware/sdma_memory_to_memory_cm4.elf <tftp_server_ip>
echo sdma_memory_to_memory_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the example runs successfully, you can see the similar information from the terminal as below.

```
SDMA memory to memory transfer example begin.

Destination Buffer:
0       0       0       0

SDMA memory to memory transfer example finish.

Destination Buffer:
1       2       3       4
```

scatter_gather

Description

The SDMA scatter gather example is a simple demonstration program that uses the SDK software. It executes several shots transfer from source buffer to destination buffer using the SDK SDMA drivers. The purpose of this example is to show how to use the SDMA and to provide a scatter gather example for debugging and further development.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

To use the example, please mind the necessary changes.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "sdma_scatter_gather.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r sdma_scatter_gather_cm4.elf -l /lib/firmware/sdma_scat-
ter_gather_cm4.elf <tftp_server_ip>
echo sdma_scatter_gather_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the example runs successfully, you can see the similar information from the terminal as below.

```
SDMA scatter_gather transfer example begin.


Destination Buffer:
0       0       0       0       0       0       0       0       0
0       0       0       0       0       0       0


SDMA scatter_gather transfer example finish.


Destination Buffer:
0       1       2       3       4       5       6       7       8
9       10       11       12       13       14       15
~~~~~~~~~~~~~~~~~~~~
```

8.4.10 sema4

Description

The sema4 uboot example shows how to use SEMA4 driver to lock and unlock a sema gate, the notification IRQ is also demonstrated in this example. This example should work together with uboot. This example runs on Cortex-M core, the uboot runs on the Cortex-A core.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can only be executed from U-Boot.

1. Execute from U-Boot

```
setenv "sema4_uboot.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

Output

Follow the output log, lock and unlock the sema4 gate in uboot. The whole log:

```
SEMA4 uboot example start
Lock sema4 gate in uboot using:
 > mw.b 0x30ac0000 1 1
Unlock sema4 gate in uboot using:
 > mw.b 0x30ac0000 0 1
SEMA4 uboot example success
```

8.4.11 tmu

tmu_1_monitor_threshold

Description

The TMU example shows how to configure TMU register to monitor and report the temperature from the temperature measurement site located on the chip.

TMU has access to temperature measurement site located on the chip. It can signal an alarm if a programmed threshold is ever exceeded.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "tmu_1_monitor_threshold.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r tmu_1_monitor_threshold_cm4.elf -l
/lib/firmware/tmu_1_monitor_threshold_cm4.elf <tftp_server_ip>
echo tmu_1_monitor_threshold_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the example runs successfully, you will see the temperature output from the terminal every time when the pre-set threshold is reached.

tmu_1_temperature_polling

Description

The TMU example shows how to configure TMU register to monitor and report the temperature from the temperature measurement site located on the chip.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "tmu_1_temperature_polling.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r tmu_1_temperature_polling_cm4.elf -l
/lib/firmware/tmu_1_temperature_polling_cm4.elf <tftp_server_ip>

echo tmu_1_temperature_polling_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the example runs successfully, you will see the temperature output from the terminal.

8.4.12 uart

auto_baudrate_detect

Description

The uart_auto_baudrate_detect example shows how to use uart auto baud rate detect feature: In this example, one uart instance connect to PC through uart. First, we should send characters a or A to board. The boars will set baud rate automatic. After baud rate has set, the board will send back all characters that PC send to the board.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "iuart_auto_baudrate_detect.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r iuart_auto_baudrate_detect_cm4.elf -l
/lib/firmware/iuart_auto_baudrate_detect_cm4.elf <tftp_server_ip>

echo iuart_auto_baudrate_detect_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

Set any baud rate in your terminal, and send character a or A to board, then

When the demo runs successfully, the log would be seen on the debug terminal like:

```
UART has detect one character A

Baud rate has been set automatic!

Board will send back received characters
```

idle_detect_sdma_transfer

Description

The uart_idle_detect_sdma example shows how to use uart driver in sdma way:

In this example, one uart instance connect to PC through uart, the board will send back all characters that PC send to the board.

Uart will receive 8 characters every time, but if the character is less then 8, the idle line interrupt will generate, and abort the SDMA receive operation, and send out the received characters.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

This example is **not** imported for FS 8MM OSM-SF.

To use the example, please mind the necessary changes.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "iuart_idle_detect_sdma_transfer.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r iuart_idle_detect_sdma_transfer_cm4.elf -l
/lib/firmware/iuart_idle_detect_sdma_transfer_cm4.elf
<tftp_server_ip>
echo iuart_idle_detect_sdma_transfer_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
Uart sdma transfer example!
Uart will receive 8 charactes every time, if less characters were
received,
Uart will generate the idle line detect interrupt, SDMA receive
operation will be aborted.
Board will send the received characters out.
Now please input:
```

hardware_flow_control

Description

The uart_hardware_flow_control example shows how to send data via UART to itself. The hardware flow control detects and avoids overflows. This functionality is realized by two additional pins

- Request To Send (RTS)
- Clear To Send (CTS)

The CTS pin belongs to the transmitter and checks the RTS pin of the receiver. If the CTS pin asserts, the transmitter starts to send data.

This example works only for PicoCoreBBDSI rev. 1.30 and higher.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

This example is **not** imported for FS 8MM OSM-SF.

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.4
UART_B_TXD	J1_28		J10_5
UART_B_RXD	J1_26		J10_3
UART_B_CTS	J1_24		J10_6
UART_B_RTS	J1_22		J10_4

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "iuart_hardware_flow_control.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r iuart_hardware_flow_control_cm4.elf -l
/lib/firmware/iuart_hardware_flow_control_cm4.elf <tftp_server_ip>

echo iuart_hardware_flow_control_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

If the execution is successful, the following message is displayed in the terminal window

```
This is UART hardware flow control example on one board.

This example will send data to itself and will use hardware flow
control to avoid the overflow. Please make sure you make the cor-
rect line connection. Basically, the connection is:

UART_TX   --    UART_RX

UART_RTS  --    UART_CTS

Data matched! Transfer successfully.
```

interrupt

Description

The uart_functioncal_interrupt example shows how to use uart driver functional API to receive data with interrupt method:

In this example, one uart instance connect to PC through uart, the board will send back all characters that PC send to the board.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "iuart_interrupt.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r iuart_interrupt_cm4.elf -l /lib/firmware/iuart_inter-
rupt_cm4.elf <tftp_server_ip>
echo iuart_interrupt_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
Uart functional interrupt example
Board receives characters then sends them out
Now please input:
```

interrupt_rb_transfer

Description

The uart_interrupt_ring_buffer example shows how to use uart driver in interrupt way with RX ring buffer enabled:

In this example, one uart instance connect to PC through uart, the board will send back all characters that PC send to the board.

The example echoes every 8 characters, so input 8 characters every time.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "iuart_interrupt_rb_transfer.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r iuart_interrupt_rb_transfer_cm4.elf -l
/lib/firmware/iuart_interrupt_rb_transfer_cm4.elf <tftp_server_ip>

echo iuart_interrupt_rb_transfer_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
UART RX ring buffer example

Send back received data

Echo every 8 bytes
```

interrupt_transfer

Description

The uart_interrupt example shows how to use uart driver in interrupt way:

In this example, one uart instance connect to PC through uart, the board will send back all characters that PC send to the board.

The example echoes every 8 characters, so input 8 characters every time.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "iuart_interrupt_transfer.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r iuart_interrupt_transfer_cm4.elf -l
/lib/firmware/iuart_interrupt_transfer_cm4.elf <tftp_server_ip>

echo iuart_interrupt_transfer_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
Uart interrupt example

Board receives 8 characters then sends them out

Now please input:
```

polling

Description

The uart_polling example shows how to use uart driver in polling way:

In this example, one uart instance connect to PC through uart, the board will send back all characters that PC send to the board.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "iuart_polling.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r iuart_polling_cm4.elf -l
/lib/firmware/iuart_polling_cm4.elf <tftp_server_ip>
echo iuart_polling_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
Uart polling example
Board will send back received characters
```

sdma_transfer

Description

The uart_sdma example shows how to use uart driver in sdma way:

In this example, one uart instance connect to PC through uart, the board will send back all characters that PC send to the board.

The example echoes every 8 characters, so input 8 characters every time.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

This example is **not** imported for FS 8MM OSM-SF.

To use the example, please mind the necessary changes.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "iuart_sdma_transfer.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r iuart_sdma_transfer_cm4.elf -l
/lib/firmware/iuart_sdma_transfer_cm4.elf <tftp_server_ip>

echo iuart_sdma_transfer_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
Uart interrupt example

Board receives 8 characters then sends them out

Now please input:


When you input 8 characters, the system will echo it by UART.
```

8.4.13 wdog

Description

The WDOG Example project is to demonstrate usage of the KSDK wdog driver. In this example,implemented to test the wdog.

Please notice that because WDOG control registers are write-once only. And for the field WDT, once software performs a write "1" operation to this bit, it can not be reset/cleared until the next POR, this bit does not get reset/ cleared due to any system reset. So the WDOG_Init function can be called only once after power reset when WDT set, and the WDOG_Disable function can be called only once after reset.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "wdog01.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r wdog01_cm4.elf -l /lib/firmware/wdog01_cm4.elf
<tftp_server_ip>

echo wdog01_cm4.elf > /sys/class/remoteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

When the demo runs successfully, the log would be seen on the debug terminal like:

```
******** System Start ********
System reset by: Power On Reset!

- 3.Test the WDOG refresh function by using interrupt.
--- wdog Init done---

WDOG has be refreshed!
WDOG has be refreshed!
WDOG has be refreshed!
WDOG has be refreshed!
WDOG has be refreshed!
...
```

8.5 mulitcore_examples

8.5.1 rpmsg_lite_pingpong_rtos_linux_remote

Description

The Multicore RPMsg-Lite pingpong RTOS project is a simple demonstration program that uses the MCUXpresso SDK software and the RPMsg-Lite library and shows how to implement the inter-core communicaton between cores of the multicore system. The primary core releases the secondary core from the reset and then the inter-core communication is established. Once the RPMsg is initialized and endpoints are created the message exchange starts, incrementing a virtual counter that is part of the message payload. The message pingpong finishes when the counter reaches the value of 100. Then the RPMsg-Lite is deinitialized and the procedure of the data exchange is repeated again.

Shared memory usage

This multicore example uses the shared memory for data exchange. The shared memory region is defined and the size can be adjustable in the linker file. The shared memory region start address and the size have to be defined in linker file for each core equally. The shared memory start address is then exported from the linker to the application.

Modifications made

board.h

```
#define VDEV0_VRING_BASE (0x50000000U)
```

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Adjust the RPMsg addresses according to the RAM size.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "rpmsg_lite_pingpong_rtos_linux_remote.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

then wait for Linux OS to finish booting. Log in, and type

```
modprobe imx_rpmsg_pingpong
```

to load the pingpong Linux side module.

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r rpmsg_lite_pingpong_rtos_linux_remote_cm4.elf -l
/lib/firmware/rpmsg_lite_pingpong_rtos_linux_remote_cm4.elf
<tftp_server_ip>

echo rpmsg_lite_pingpong_rtos_linux_remote_cm4.elf >
/sys/class/remoteproc/remoteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state

modprobe imx_rpmsg_pingpong
```

Output

```
RPMSG Ping-Pong FreeRTOS RTOS API Demo...
RPMSG Share Base Addr is 0xb5000000
```

During boot the Kernel,the ARM Cortex-M4 terminal displays the following information:

```
Link is up!
Nameservice announce sent.
```

After the Linux RPMsg pingpong module was installed, the ARM Cortex-M4 terminal displays the following information:

```
Waiting for ping...
Sending pong...
Waiting for ping...
Sending pong...
Waiting for ping...
Sending pong...
......
Waiting for ping...
Sending pong...
Ping pong done, deinitializing...
Looping forever...
```

The Cortex-A terminal displays the following information:

```
get 1 (src: 0x1e)
get 3 (src: 0x1e)
......
get 99 (src: 0x1e)
get 101 (src: 0x1e)
```

8.5.2 rpmsg_lite_str_echo_rtos

Description

The Multicore RPMsg-Lite string echo project is a simple demonstration program that uses the MCUXpresso SDK software and the RPMsg-Lite library and shows how to implement the inter-core communicaton between cores of the multicore system.

It works with Linux RPMsg master peer to transfer string content back and forth. The name service handshake is performed first to create the communication channels. Next, Linux OS waits for user input to the RPMsg virtual tty. Anything which is received is sent to M4. M4 displays what is received, and echoes back the same message as an acknowledgement. The tty reader on the Linux side can get the message, and start another transaction. The demo demonstrates RPMsg's ability to send arbitrary content back and forth.

The maximum message length supported by RPMsg is now 496 bytes. String longer than 496 will be divided by virtual tty into several messages.

Shared memory usage

This multicore example uses the shared memory for data exchange. The shared memory region is defined and the size can be adjustable in the linker file. The shared memory region start address and the size have to be defined in linker file for each core equally. The shared memory start address is then exported from the linker to the application.

Modifications made

board.h

```
#define VDEV0_VRING_BASE (0x50000000U)
```

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

Adjust the RPMsg addresses according to the RAM size.

Example Execution

This example can be executed from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "rpmsg_lite_str_echo_rtos_remote_cm4.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

then wait for Linux OS to finish booting. Log in, and type

```
modprobe imx_rpmsg_tty
```

to load the pingpong Linux side module.

Run

```
echo test > /dev/ttyRPMSG30
```

to show the output of the RPMsg-Lite str echo demo in the terminal window

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r rpmsg_lite_str_echo_rtos_remote_cm4.elf -l
/lib/firmware/rpmsg_lite_str_echo_rtos_remote_cm4.elf
<tftp_server_ip>
echo rpmsg_lite_str_echo_rtos_remote_cm4.elf > /sys/class/re-
moteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
modprobe imx_rpmsg_tty
```

Output

```
RPMSG String Echo FreeRTOS RTOS API Demo...


Nameservice sent, ready for incoming messages...
```

After the Linux RPMsg tty module was installed, the ARM Cortex-M4 terminal displays the following information:

```
Get Message From Master Side : "hello world!" [len : 12]
```

After the user write into the ttyRPMSG –device the Cortex-M4 terminal displays the following information:

```
Get Message From Master Side : "test" [len : 4]
Get New Line From Master Side
```

8.6 rtos_examples

8.6.1 freertos_event

Description

This document explains the freertos_event example. It shows how task waits for an event (defined set of bits in event group). This event can be set by any other process or interrupt in the system.

The example application creates three tasks. Two write tasks write_task_1 and write_task_2 continuously setting event bit 0 and bit 1.

Read_task is waiting for any event bit and printing actual state on console. Event bits are automatically cleared after read task is entered.

Three possible states can occurre:

Both bits are set.

Bit B0 is set.

Bit B1 is set.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be execution from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "freertos_event.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r freertos_event_cm4.elf -l /lib/firmware/freer-
tos_event_cm4.elf <tftp_server_ip>

echo freertos_event_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

After the board is flashed the Tera Term will start printing the state of event bits.

```
Bit B1 is set.

Bit B0 is set.

Bit B1 is set.

Bit B0 is set.

Bit B1 is set.

. . .
```

8.6.2 freertos_generic

Description

This document explains the freertos_generic example. It is based on code FreeRTOS documentation from http://www.freertos.org/Hardware-independent-RTOS-example.html. It shows combination of several tasks with queue, software timer, tick hook and semaphore.

The example application creates three tasks. The prvQueueSendTask periodically sending data to xQueue queue. The prvQueueReceiveTask is waiting for incoming message and counting number of received messages. Task prvEventSemaphoreTask is waiting for xEventSemaphore semaphore given from vApplicationTickHook. Tick hook give semaphore every 500 ms.

Other hook types used for RTOS and resource statistics are also demonstrated in example:

- vApplicationIdleHook
- vApplicationStackOverflowHook
- vApplicationMallocFailedHook

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be execution from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "freertos_generic.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r freertos_generic_cm4.elf -l /lib/firmware/freer-
tos_generic_cm4.elf <tftp_server_ip>

echo freertos_generic_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

After the board is flashed the Tera Term will start periodically printing the state of generic example.

```
Event task is running.

Receive message counter: 1.

Receive message counter: 2.

Receive message counter: 3.

Receive message counter: 4.

Receive message counter: 5.

Receive message counter: 6.

Receive message counter: 7.

. . .
```

8.6.3 freertos_hello

Description

The Hello World project is a simple demonstration program that uses the SDK UART driver in combination with FreeRTOS. The purpose of this demo is to show how to use the debug console and to provide a simple project for debugging and further development.

The example application creates one task called hello_task. This task print "Hello world." Message via debug console utility and suspend itself.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be execution from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "freertos_hello.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r freertos_hello_cm4.elf -l /lib/firmware/freer-
tos_hello_cm4.elf <tftp_server_ip>
echo freertos_hello_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

After the board is flashed the Tera Term will print "Hello world" message on terminal.

```
Hello world.
```

8.6.4 freertos_mutex

Description

This document explains the freertos_mutex example. It shows how mutex manage access to common resource (terminal output).

The example application creates two identical instances of write_task. Each task will lock the mutex before printing and unlock it after printing to ensure that the outputs from tasks are not mixed together.

The test_task accept output message during creation as function parameter. Output message have two parts. If xMutex is unlocked, the write_task_1 acquire xMutex and print first part of message. Then rescheduling is performed. In this moment scheduler check if some other task could run, but second task write_task+_2 is blocked because xMutex is already locked by first write task. The first write_task_1 continue from last point by printing of second message part. Finaly the xMutex is unlocked and second instance of write_task_2 is executed.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Changes needed

None

Example Execution

This example can be execution from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "freertos_mutex.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r freertos_mutex_cm4.elf -l /lib/firmware/freertos_mu-
tex_cm4.elf <tftp_server_ip>

echo freertos_mutex_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

After the board is flashed the Tera Term will start periodically printing strings synchronized by mutex.

```
"ABCD | EFGH"

"1234 | 5678"

"ABCD | EFGH"

"1234 | 5678"

…
```

8.6.5 freertos_queue

Description

This document explains the freertos_queue example. This example introduce simple logging mechanism based on message passing.

Example could be devided in two parts. First part is logger. It contain three tasks:

log_add().....Add new message into the log. Call xQueueSend function to pass new message into message

queue.log_init()....Initialize logger (create logging task and message queue log_queue).

log_task()....Task responsible for printing of log output.

Second part is application of this simple logging mechanism. Each of two tasks write_task_1 and write_task_2 print 5 messages into log.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Add an empty_rsc_table.c file.

Changes needed

None

Example Execution

This example can be execution from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "freertos_queue.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r freertos_queue_cm4.elf -l /lib/firmware/freertos_queue_cm4.elf <tftp_server_ip>
echo freertos_queue_cm4.elf > /sys/class/remoteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

After the board is flashed the Tera Term will show debug console output.

```
Log 0: Task1 Message 0
Log 1: Task2 Message 0
Log 2: Task1 Message 1
Log 3: Task2 Message 1
. . .
Log9:  Task2 Message 4
```

8.6.6 freertos_sem

Description

This document explains the freertos_sem example, what to expect when running it and a brief introduction to the API. The freertos_sem example code shows how semaphores works. Two different tasks are synchronized in bilateral rendezvous model.

The example uses four tasks. One producer_task and three consumer_tasks. The producer_task starts by creating of two semaphores (xSemaphore_producer and xSemaphore_consumer). These semaphores control access to virtual item. The synchronization is based on bilateral rendezvous pattern. Both of consumer and producer must be prepared to enable transaction.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Add an empty_rsc_table.c file.

Changes needed

None

Example Execution

This example can be execution from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "freertos_sem.bin";

setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"

run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r freertos_sem_cm4.elf -l /lib/firmware/freertos_sem_cm4.elf <tftp_server_ip>
echo freertos_sem_cm4.elf > /sys/class/remoteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

After the board is flashed the Tera Term will show debug console output.

```
Producer_task created.
Consumer_task 0 created.
Consumer_task 1 created.
Consumer_task 2 created.
Consumer number: 0
Consumer 0 accepted item.
Consumer number: 1
Consumer number: 2
Producer released item.
Consumer 0 accepted item.
Producer released item.
Consumer 1 accepted item.
Producer released item.
Consumer 2 accepted item.
. . .
```

8.6.7 freertos_sem_static

This document explains the freertos_sem example, what to expect when running it and a brief introduction to the API. The freertos_sem example code shows how semaphores works. Two different tasks are synchronized in bilateral rendezvous model.

The example uses four tasks. One producer_task and three consumer_tasks. The producer_task starts by creating of two semaphores (xSemaphore_producer and xSemaphore_consumer). These semaphores control access to virtual item. The synchronization is based on bilateral rendezvous pattern. Both of consumer and producer must be prepared to enable transaction.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Add an empty_rsc_table.c file.

Changes needed

None

Example Execution

This example can be execution from U-Boot or in Linux.

3. Execute from U-Boot

```
setenv "freertos_sem_static.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

4. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r freertos_sem_static_cm4.elf -l /lib/firmware/freer-
tos_sem_static_cm4.elf <tftp_server_ip>
echo freertos_sem_static_cm4.elf > /sys/class/remoteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

After the board is flashed the Tera Term will show debug console output.

```
Producer_task created.
Consumer_task 0 created.
Consumer_task 1 created.
Consumer_task 2 created.
Consumer number: 0
Consumer 0 accepted item.
Consumer number: 1
Consumer number: 2
Producer released item.
Consumer 0 accepted item.
Producer released item.
Consumer 1 accepted item.
Producer released item.
Consumer 2 accepted item.
. . .
```

8.6.8 freertos_swtimer

Description

This document explains the freertos_swtimer example. It shows usage of software timer and its callback.

The example application creates one software timer SwTimer. The timer's callback SwTimer-Callback is periodically executed and text "Tick." is printed to terminal.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Add an empty_rsc_table.c file.

Changes needed

None

Example Execution

This example can be execution from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "freertos_swtimer.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra "clk-imx8mm.mcore_booted=1"
```

Then run the following commands in Linux:

```
tftp -g -r freertos_swtimer_cm4.elf -l /lib/firmware/freer-
tos_swtimer_cm4.elf <tftp_server_ip>

echo freertos_swtimer_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware

echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

After the board is flashed the Tera Term will show output message.

```
Tick.

Tick.

Tick.

. . .
```

8.6.9 freertos_tickless

Description

This document explains the freertos_tickless example. It shows how the CPU enters the sleep mode and then it is woken up either by expired time delay using low power timer module or by external interrupt caused by a user defined button.

Modifications made

MIMX8MM6xxxLZ_ram.ld

```
m_data2 (RW) : ORIGIN = 0x55500000, LENGTH = 0x01000000
```

Add an empty_rsc_table.c file.

Changes needed

None

Example Execution

This example can be execution from U-Boot or in Linux.

1. Execute from U-Boot

```
setenv "freertos_tickless.bin";
setenv m4 "tftp ${m4_file}; cp.b $loadaddr 0x007E0000 $filesize;
bootaux 0x007E0000"
run m4
```

2. Execute from Linux

First set in U-Boot the following variable, if already not done:

```
setenv extra “clk-imx8mm.mcore_booted=1”
```

Then run the following commands in Linux:

```
tftp -g -r freertos_tickless_cm4.elf -l /lib/firmware/freer-
tos_tickless_cm4.elf <tftp_server_ip>
echo freertos_tickless_cm4.elf > /sys/class/remoteproc/re-
moteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

Output

After the board is flashed the Tera Term will show debug console output.

```
0
5000
10000
15000
20000
25000
30000
. . .
```

9 Appendix

List of Figures

List of Tables

Third Party Agreement from Real Time Engineers Ltd.

Any FreeRTOS source code, whether modified or in its original release form, or whether in whole or in part, can only be distributed by you under the terms of version 2 of the GNU General Public License plus this exception. An independent module is a module which is not derived from or based on FreeRTOS.

Clause 1: Linking FreeRTOS with other modules is making a combined work based on FreeRTOS. Thus, the terms and conditions of the GNU General Public License V2 cover the whole combination.

As a special exception, the copyright holders of FreeRTOS give you permission to link FreeRTOS with independent modules to produce a statically linked executable, regardless of the license terms of these independent modules, and to copy and distribute the resulting executable under terms of your choice, provided that you also meet, for each linked independent module, the terms and conditions of the license of that module. An independent module is a module which is not derived from or based on FreeRTOS.

Clause 2: FreeRTOS may not be used for any competitive or comparative purpose, including the publication of any form of run time or compile time metric, without the express permission of Real Time Engineers Ltd. (this is the norm within the industry and is intended to ensure information accuracy).

Important Notice

The information in this publication has been carefully checked and is believed to be entirely accurate at the time of publication. F&S Elektronik Systeme assumes no responsibility, however, for possible errors or omissions, or for any consequences resulting from the use of the information contained in this documentation.

F&S Elektronik Systeme reserves the right to make changes in its products or product specifications or product documentation with the intent to improve function or design at any time and without notice and is not required to update this documentation to reflect such changes.

F&S Elektronik Systeme makes no warranty or guarantee regarding the suitability of its products for any particular purpose, nor does F&S Elektronik Systeme assume any liability arising out of the documentation or use of any product and specifically disclaims any and all liability, including without limitation any consequential or incidental damages.

Products are not designed, intended, or authorised for use as components in systems intended for applications intended to support or sustain life, or for any other application in which the failure of the product from F&S Elektronik Systeme could create a situation where personal injury or death may occur. Should the Buyer purchase or use a F&S Elektronik Systeme product for any such unintended or unauthorised application, the Buyer shall indemnify and hold F&S Elektronik Systeme and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, expenses, and reasonable attorney fees arising out of, either directly or indirectly, any claim of personal injury or death that may be associated with such unintended or unauthorised use, even if such claim alleges that F&S Elektronik Systeme was negligent regarding the design or manufacture of said product.