FreeRTOS on FSiMX8MM Boards

Manual on how to use/configuring the software

Version 1.1
(2021-07-19)

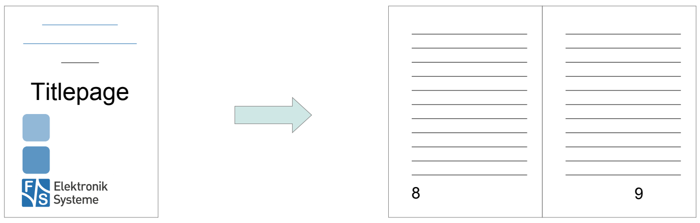
About This Document

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

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 1.3
SPI_B_MOSI / LED	J1_60		J11_6

1.1.2 ECSPI

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

1.1.5 UART_B

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.3
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

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-9-2020-q2-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:

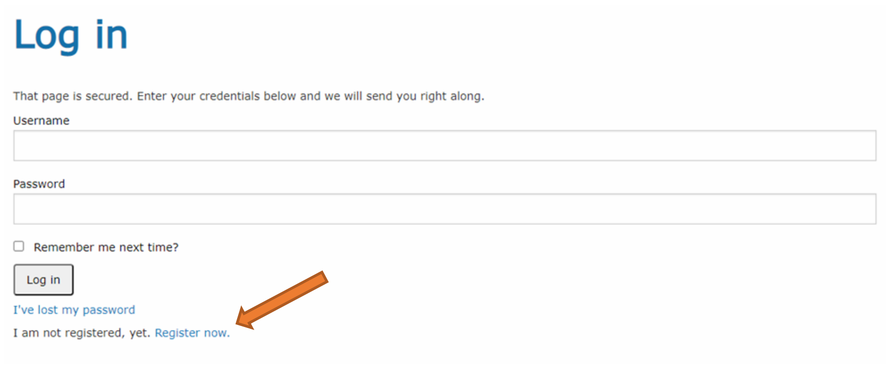
```
export ARMGCC_DIR=/usr/local/arm/gcc-arm-none-eabi-${version}
```

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*).



Log in

That page is secured. Enter your credentials below and we will send you right along.

Username

Password

Remember me next time?

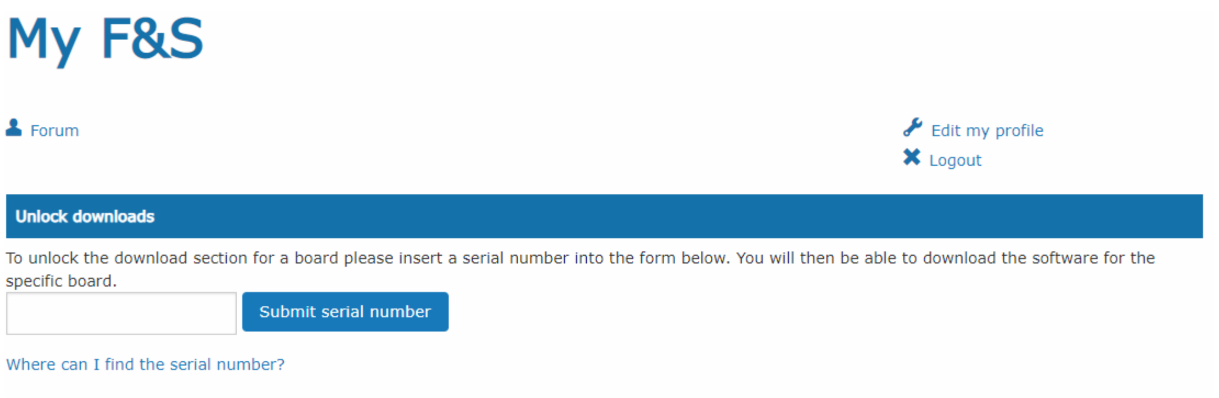
I've lost my password

I am not registered, yet. Register now.

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.



My F&S

Unlock downloads

To unlock the download section for a board please insert a serial number into the form below. You will then be able to download the software for the specific board.

Where can I find the serial number?

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-2.9.1-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-2.9.1-<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.).

Installation

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.

The provided NBoot version 2021.07 or higher must be installed. It contains needed changes for the Cortex-M4.

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`
`nboot-fsimx8mm-<v>.fs`
	Precompiled examples for PicoCoreMX8MM
	Required NBoot for this release

`sources/`	**Source packages**
`freertos-sdk-2.9.1-fsimx8mm-V2021.07.tar.bz2`	FreeRTOS source
`toolchain/`	**Cross-compilation toolchain**

Directory/File	Description
`gcc-arm-none-eabi-9-2020-q2-update-x86_64-linux.tar.bz2`	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-2.9.1-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-2.9.1-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-2.9.1-fsimx8mm-V<year>.<month>.tar.bz2
```

Please install the provided NBoot located in `binaries/nboot-fsimx8mm-<v>.fs`

4.4 Installation of the GCC embedded toolchain

The examples can be built with the GCC embedded toolchain (gcc-arm-none-eabi-9-2020-q2), 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 provides a necessary patch for the Linux kernel. This change is crucial. The patch is independent from the release type, it must be applied for fsimx8mm-Y2021.04 and for fsimx8mm-B2021.06. For newer fsimx8mm releases the patch is not necessary anymore. Please make sure to install it according to the used one.

4.5.1 Additional binaries

Besides the binaries of the respective FreeRTOS example in the `binaries/` directory, the required NBoot version 2021.07 or higher can be found here. It contains additional fixes for the Cortex-M4. This version must be installed, otherwise certain examples will not work correctly. For further information on how to install/update the NBoot, please refer to the document "`LinuxOnFSBoards`".

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 make command the output binaries or images can be found here in their specific $boardname-directory.
build	After you have run the make command, this directory contains the .bin, .elf, .hex, .map and object files for each example.
CMakeFiles	Contains CMake-specific files. Normally you don't have to change anything in here.
CMSIS	Contains the Cortex Microcontroller Software Interface Standard (CMSIS) library.
devices	Contains socket specific files and drivers.
doc	Contains the original documentation by NXP.
examples/	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 Microcontroller 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.
not_tested	Contains examples that have not been tested 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. For further information refer to the porting_readme.txt located at not_tested/<soc>/.
rtos	Contains the operating system freertos.
tools	Contains different tools needed for the building process.

Table 2: Description of the directory structure

5 Configuration for Cortex-M4 usage

5.1 Changes regarding official 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:

Run

```
setenv m4_file <example_name>
```

to set the name of the example you want to load.

Run

```
run m4
```

5.2 Using bootaux

Simple start

Using the auxiliary core can be achieved by using the following command line inside of the U-Boot environment:

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

This will load an image defined by *m4_file* via tftp to your board, move it to the TCM and start the auxiliary core.

6 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:

6.1 Adjusting the right UART Console

If the PicoCoreBBDSI REV >= 1.20 is used, this step can be ignored.

Users that use the PicoCoreBBDSI REV1.10 baseboard need to make a few changes in every example they may run since there are major differences in the Output of those ports between revision 1.10 and 1.20. since the examples are ported to run on revision 1.20.

The changes are basically the same in every example, except for those that show UART functionality. UART examples have to be modified further by changing the ‘4’ in every mention of UART into ‘3’. For all other examples it is just a few lines that need minor changes that are necessary to get an output on the Debug Console.

The following list shows all the changes necessary for **all examples**:

board.c

```
CLOCK_EnableClock(kCLOCK_Uart4);
```

Needs to be changed to:

```
CLOCK_EnableClock(kCLOCK_Uart3);
```

board.h

```
#define BOARD_DEBUG_UART_BASEADDR UART4_BASE
#define BOARD_DEBUG_UART_INSTANCE 4U
#define BOARD_DEBUG_UART_CLK_FREQ
\
    CLOCK_GetPllFreq(kCLOCK_SystemPll1Ctrl) /
(CLOCK_GetRootPreDivider(kCLOCK_RootUart4)) / \
        (CLOCK_GetRootPostDivider(kCLOCK_RootUart4)) / 10
#define BOARD_UART_IRQ UART4_IRQn
#define BOARD_UART_IRQ_HANDLER UART4_IRQHandler
```

Needs to be changed to:

```
#define BOARD_DEBUG_UART_BASEADDR UART3_BASE
#define BOARD_DEBUG_UART_INSTANCE 3U
#define BOARD_DEBUG_UART_CLK_FREQ
\
    CLOCK_GetPllFreq(kCLOCK_SystemPll1Ctrl) /
(CLOCK_GetRootPreDivider(kCLOCK_RootUart3)) / \

        (CLOCK_GetRootPostDivider(kCLOCK_RootUart3)) / 10

#define BOARD_UART_IRQ UART3_IRQn

#define BOARD_UART_IRQ_HANDLER UART3_IRQHandler
```

clock_config.c

```
CLOCK_SetRootMux(kCLOCK_RootUart3,
kCLOCK_UartRootmuxSysPll1Div10); /* Set UART source to SysPLL1
Div10 80MHZ */

    CLOCK_SetRootDivider(kCLOCK_RootUart3, 1U, 1U);
/* Set root clock to 80MHZ/ 1= 80MHZ */
```

Needs to be changed to:

```
CLOCK_SetRootMux(kCLOCK_RootUart3,
kCLOCK_UartRootmuxSysPll1Div10); /* Set UART source to SysPLL1
Div10 80MHZ */

    CLOCK_SetRootDivider(kCLOCK_RootUart3, 1U, 1U);
/* Set root clock to 80MHZ/ 1= 80MHZ */
```

pin_muc.c

```
IOMUXC_SetPinMux(IOMUXC_UART4_RXD_UART4_RX, 0U);

    IOMUXC_SetPinConfig(IOMUXC_UART4_RXD_UART4_RX,

                        IOMUXC_SW_PAD_CTL_PAD_DSE(6U) |

                        IOMUXC_SW_PAD_CTL_PAD_FSEL(2U));

    IOMUXC_SetPinMux(IOMUXC_UART4_TXD_UART4_TX, 0U);

    IOMUXC_SetPinConfig(IOMUXC_UART4_TXD_UART4_TX,

                        IOMUXC_SW_PAD_CTL_PAD_DSE(6U) |

                        IOMUXC_SW_PAD_CTL_PAD_FSEL(2U));
```

Needs to be changed to:

```
IOMUXC_SetPinMux(IOMUXC_UART3_RXD_UART3_RX, 0U);

    IOMUXC_SetPinConfig(IOMUXC_UART3_RXD_UART3_RX,

                        IOMUXC_SW_PAD_CTL_PAD_DSE(6U) |

                        IOMUXC_SW_PAD_CTL_PAD_FSEL(2U));

    IOMUXC_SetPinMux(IOMUXC_UART3_TXD_UART3_TX, 0U);

    IOMUXC_SetPinConfig(IOMUXC_UART3_TXD_UART3_TX,
                        IOMUXC_SW_PAD_CTL_PAD_DSE(6U) |
                        IOMUXC_SW_PAD_CTL_PAD_FSEL(2U));
```

6.2 Prepare.sh

This script will configure board relevant settings and create symlinks to the board specific header files. You can execute the script in your terminal by typing

```
./prepare.sh
```

and follow the instructions:

```
Choose one of the following boards for which you want to build the
examples:
efusa9x[1] picocoma9x[2]      …[…]       picocoremx8mm[5]
Enter number in []-brackets for the corresponding board: 5
Please choose the production variant V3/V4 (LPDDR4) [1] V5/V6 (DDR3L)
[2]:
Do you want a Release or Debug build?
(r/d) [default: r]: r
All set up, starting cmake...
```

If you have different DRAM sizes for your boards, just edit `prepare.sh` to your personal preferences. If you've chosen the picocoremx8mm board you'll be asked to select a module variant.

Most of the examples can be run from TCM or directly from the QSPI-flash. In future releases it will be possible to choose this in the prepare.sh script but for now only TCM is supported.

6.3 Make

The `prepare.sh` script will configure and invoke *CMake* to generate a Makefile. After this, you can run

```
make -jN
```

To build all examples located in `examples/fsimx8mm` and install the binaries to `bin/$BOARD`.

`N` is the number of cores your CPU have.

If you want to build a specific example just type

```
make -jN example_name && make install/fast
```

to build and install the binary of the chosen example.

Type

```
make help
```

for a list of possible examples for make.

By executing

```
make clean-all
```

you can clean up all build files and binaries. This will be necessary if you make changes to the `CMakeLists.txt` in the root directory of the FreeRTOS BSP or when the target module has changed i.e. V3/V4 to V5/V6. In this case, just rerunning the prepare.sh script won't be sufficient and will lead to faulty variables.

7 Adding custom boards

If you're using a custom board, you have to tell the prepare.sh script about its existence and create some configuration files (or simply copy the existing ones).

To tell the script about it, change the following lines in the prepare.sh script:

```
declare -a SUPPORTED_BOARDS=("efusa9x" "picocoma9x" "picocoremx6sx"
“picocoremx7ulp” “picocoremx8mm” "boardname")
```

where *boardname* represent the name of your board and an entry to

```
declare -a SUPPORTED_SOCS=("fsimx6sx" "fsimx6sx" "fsimx6sx" "fsimx7ulp" “fsimx6sx”
“fsimx8mm” “SOC”)
```

so the number of your *boardname* matches the number of its specific SOC.

The following files, located at examples/fsimx6sx/board_specific_files/*boardname*, are needed to successfully compile the BSP for your own board:

- *boardname*_board.c
- *boardname*_board.h
- *boardname*_pin_mux.c
- *boardname*_pin_mux.h
- *boardname*_gpio_pins.c
- *boardname*_gpio_pins.h

boardname must be replaced by the name of your board. This must be same name as uses in the *SUPPORTED_BOARDS* array used in the prepare.sh script.

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

Connect UART4 (Cortex M4) and UART1 (Cortex A53) with two serial cables (serial-to-modem) to your PC.

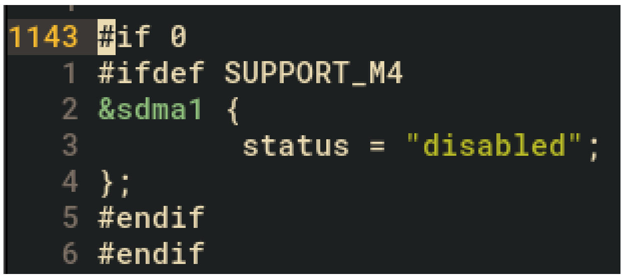
Open up two Terminals and connect the UARTs via the COM interface and the following settings:

```
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 use the SDMA examples, the device tree has to be modified so that the M4 has access to the SDMA module. However, this will also take away the SPI_A from the A53 core.

To do so, please change the #if 0 to #if 1



```
#if 0
#ifdef SUPPORT_M4
&sdma1 {
        status = "disabled";
};
#endif
#endif
```

Figure 3: Device tree entry

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

None.

Changes needed

None.

Execute binary

Run

```
setenv m4_file "cmsis_ecspi_int_loopback_transfer.bin";
run m4;
```

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
internally.
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

None.

Changes needed

To use the example, please mind the necessary changes.

Execute binary

Run

```
setenv m4_file "cmsis_ecspi_sdma_loopback_transfer.bin";
run m4;
```

to start the example.

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
internally.
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

Changes needed

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

Execute binary

Run

```
setenv m4_file "cmsis_ii2c_b2b_transfer_master.bin";
run m4;
```

to start the example.

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

Changes needed

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

Execute binary

Run

```
setenv m4_file "cmsis_ii2c_b2b_transfer_slave.bin";
run m4;
```

to start the example.

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

None.

Changes needed

None.

Execute binary

Run

```
setenv m4_file "cmsis_iuart_interrupt_transfer.bin";
run m4;
```

to start the example.

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

None.

Changes needed

To use the example, please mind the necessary changes.

Execute binary

Run

```
setenv m4_file "cmsis_iuart_sdma_transfer.bin";
run m4;
```

to start the example.

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

None.

Changes needed

None.

Execute binary

Run

```
setenv m4_file "hello_world.bin";
run m4;
```

to start the example.

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

None.

Changes needed

None.

Execute binary

Run

```
setenv m4_file "ecspi_loopback.bin";
run m4
```

to start the example.

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
internally.
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

None.

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.2
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

Execute binary

Run

```
setenv m4_file "ecspi_interrupt_b2b_transfer_master.bin";
run m4
```

to start the example.

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

None.

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.2
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

Execute binary

Run

```
setenv m4_file "ecspi_interrupt_b2b_transfer_slave.bin";
run m4
```

to start the example.

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

None.

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.2
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

Execute binary

Run

```
setenv bootauxfile "ecspi_polling_b2b_transfer_master.bin";
run m4
```

to start the example.

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

None.

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.2
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

Execute binary

Run

```
setenv bootauxfile "ecspi_polling_b2b_transfer_master.bin";
run m4
```

to start the example.

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

None.

Changes needed

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

Execute binary

Run

```
setenv m4_file "igpio_led_output.bin";
run m4
```

to start the example.

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

None.

Changes needed

None

Execute binary

Run

```
setenv m4_file "gpt_timer.bin";
run m4
```

to start the example.

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

Changes needed

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

Execute binary

Run

```
setenv m4_file "ii2c_interrupt_b2b_transfer_master.bin";
run m4
```

to start the example.

Output

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

```
When the demo runs successfully, the following message is
displayed 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

Changes needed

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

Execute binary

Run

```
setenv m4_file "ii2c_interrupt_b2b_transfer_slave.bin";
run m4
```

to start the example.

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

Changes needed

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

Execute binary

Run

```
setenv m4_file "ii2c_polling_b2b_transfer_master.bin";
run m4
```

to start the example.

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

Changes needed

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

Execute binary

Run

```
setenv m4_file "ii2c_polling_b2b_transfer_slave.bin";
run m4
```

to start the example.

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;
```

Changes needed

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

Execute binary

First run

```
setenv m4_file "ipwm.bin";
run m4
```

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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "rdc.bin";
run m4
```

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

None.

Changes needed

To use the example, please mind the necessary changes.

Execute binary

First run

```
setenv m4_file "sdma_memory_to_memory.bin";
run m4
```

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

None.

Changes needed

To use the example, please mind the necessary changes.

Execute binary

First run

```
setenv m4_file "sdma_scatter_gather.bin";

run m4
```

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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "sema4_uboot.bin";
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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "tmu_1_monitor_threshold.bin";
run m4
```

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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "tmu_1_temperature_polling.bin";
run m4
```

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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "iuart_auto_baudrate_detect.bin";
run m4
```

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

None.

Changes needed

To use the example, please mind the necessary changes.

Execute binary

First run

```
setenv m4_file "iuart_idle_detect_sdma_transfer.bin";
run m4
```

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

None.

Changes needed

Function	PCOREMX8MM Rev 1.3	PCOREMX8MX Rev 1.2	BBDSI Rev 1.3
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

Execute binary

Run

```
setenv m4_file "iuart_hardware_flow_control";
run m4
```

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
correct 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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "iuart_interrupt.bin";
run m4
```

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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "iuart_interrupt_rb_transfer.bin";
run m4
```

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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "iuart_interrupt_transfer.bin";
run m4
```

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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "iuart_polling.bin";
run m4
```

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

None.

Changes needed

To use the example, please mind the necessary changes.

Execute binary

First run

```
setenv m4_file “iuart_sdma_transfer.bin";
run m4
```

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

None.

Changes needed

None

Execute binary

First run

```
setenv m4_file "wdog01.bin";
run m4
```

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)
```

Changes needed

Adjust the RPMsg addresses according to the RAM size.

Execute binary

First run

```
setenv m4_file "rpmsg_lite_pingpong_rtos_linux_remote.bin";
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.

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)
```

Changes needed

Adjust the RPMsg addresses according to the RAM size.

Execute binary

First run

```
setenv m4_file "rpmsg_lite_str_echo_rtos_imxcm4.bin";
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

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_ecspi

Description

The freertos_ecspi_loopback demo shows how the ecspi do a loopback transfer internally in FreeRTOS. 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

None.

Changes needed

None.

Execute binary

Run

```
setenv m4_file "freertos_ecspi_loopback.bin";
run m4
```

Output

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

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


This demo is a loopback transfer test for ECSPI.
The ECSPI will connect the transmitter and receiver sections
internally.
So, there is no need to connect the MOSI and MISO pins.


FreeRTOS ECSPI loopback test pass!
```

8.6.2 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

None

Changes needed

None

Execute binary

Run

```
setenv m4_file "freertos_event.bin";
run m4
```

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.
. . .
```

8.6.3 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

None.

Changes needed

None

Execute binary

Run

```
setenv m4_file "freertos_generic.bin";
run m4
```

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.
. . .
```

8.6.4 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

None.

Changes needed

None

Execute binary

Run

```
setenv m4_file "freertos_hello.bin";
run m4
```

Output

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

```
Hello world.
```

8.6.5 freertos_i2c

Description

This example shows how to send and receive data board to board via the I2C driver in FreeRTOS. One Board acts as the master and the other as the slave. FreeRTOS tasks will be created according to the role (Master or Slave) of the board. Single board functionality isn't yet implemented by NXP.

Modifications made

None.

Changes needed

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

Execute binary

Run

```
setenv m4_file "freertos_i2c.bin";
run m4
```

Output

Upon successful execution, the terminal window displays the following message:

Master side

```
==FreeRTOS I2C example start.==
This example use two boards to connect with one as master and
another as slave.


Master will send data :
0x 0  0x 1  0x 2  0x 3  0x 4  0x 5  0x 6  0x 7
…
0x18  0x19  0x1a  0x1b  0x1c  0x1d  0x1e 0x1f
Master received data :
<Same as above>


End of FreeRtos I2C example.
```

Slave side

```
==FreeRTOS I2C example start.==
This example use two boards to connect with one as master and
another as slave.
I2C slave transfer completed successfully.


Slave received data :
<Same as above>




End of FreeRTOS I2C example.
```

8.6.6 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

None.

Changes needed

None

Execute binary

Run

```
setenv m4_file "freertos_mutex.bin";

run m4
```

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"

…
```

8.6.7 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

None.

Changes needed

None

Execute binary

Run

```
setenv m4_file "freertos_queue.bin";
run m4
```

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
```

8.6.8 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

None.

Changes needed

None

Execute binary

Run

```
setenv m4_file "freertos_sem.bin";

run m4
```

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.
. . .
```

8.6.9 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 SwTimerCallback is periodically executed and text "Tick." is printed to terminal.

Modifications made

None.

Changes needed

None

Execute binary

Run

```
setenv m4_file "freertos_swtimer.bin";
run m4
```

Output

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

```
Tick.
Tick.
Tick.
. . .
```

8.6.10 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

None.

Changes needed

None

Execute binary

Run

```
setenv bootauxfile "freertos_tickless.bin";
run m4
```

Output

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

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

8.6.11 freertos_uart

Description

The UART example for FreeRTOS demonstrates the possibility to use the UART driver in the RTOS. The example uses single instance of UART IP and writes string into, then reads back chars. After every 4B received, these are sent back on UART.

Modifications made

None.

Changes needed

None

Execute binary

Run

```
setenv m4_file "freertos_uart.bin";
run m4
```

Output

You will see the welcome string printed out on the console. You can send characters to the console back and they will be printed out onto console in a group of 4 characters.

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